

Apunte de Teoría de Lenguajes

Tomás C.

Julio, 2022



Acerca de este apunte

Este es un compilado con los conceptos teóricos de la materia “Teoría de Lenguajes” (a.k.a. *TLeng*), escrito en base a las clases teóricas y prácticas del primer cuatrimestre de 2022, con algunos añadidos de la bibliografía de la materia.

La parte teórica de la materia en dicha cursada estuvo a cargo de Julio Jacobo y Verónica Becher (profesores), y la parte práctica de Ariel Arbiser (jefe de trabajos prácticos), mientras que los ayudantes de la cursada fueron Leonardo Cremona, Elisa Orduna, Natalia Pesaresi, Sabrina Silvero y Manuel Panichelli.

Esto no pretende ser un resumen de los contenidos de la materia en mayor medida de lo que las diapositivas de las clases son un resumen de los mismos; sino que la idea de este apunte es ser una recopilación centralizada de lo estudiado en clase, con el objetivo de *tener todo en un mismo lugar*, y disponer de él en el momento del examen final (que, al menos en esta ocasión, es *a libro abierto*).

En el caso de que este apunte sea utilizado por otras personas, me veo en la obligación de advertir que puede contener errores conceptuales, errores de redacción, errores de *típeo*, errores de ortografía o gramática, errores adentro de errores, etcétera.

El orden de los contenidos está ligeramente alterado respecto al utilizado en las clases, de acuerdo a mi criterio personal. Cada sección corresponde aproximadamente a un tema visto en la materia (aunque no todos se estudiaron en igual profundidad, de ahí las diferencias en longitud), y al final de cada sección se encuentran las clases utilizadas como fuentes de información. A su vez, al final del documento están las referencias bibliográficas generales.

Por último, los resultados marcados con el símbolo † fueron explicados (o al menos incluidos en las diapositivas de las clases) como material *adicional* y/u *opcional*.

Título completo: *Apunte teórico para el examen final de Teoría de Lenguajes*.

Autor: Tomás C.

Fecha de finalización: 29 de julio de 2022.

Imagen de portada: *Spyro, el dragón*.

Índice

1. Lenguajes y Gramáticas	1
1.1. Repaso de relaciones	1
1.1.1. Composición	1
1.1.2. Clausuras	1
1.2. Lenguajes	2
1.2.1. Clausuras	2
1.3. Gramáticas	3
1.3.1. Derivaciones	3
1.3.2. Árboles de derivación	4
1.4. Clasificación de gramáticas	5
1.4.1. Gramáticas regulares (tipo 3)	5
1.4.2. Gramáticas libres de contexto (tipo 2)	5
1.4.3. Gramáticas dependientes del contexto (tipo 1)	6
1.4.4. Gramáticas sin restricciones (tipo 0)	6
1.5. Fuentes	6
2. Autómatas Finitos	7
2.1. Autómatas Finitos Determinísticos	7
2.2. Autómatas Finitos No Determinísticos	8
2.2.1. Autómatas Finitos no Determinísticos con transiciones λ	9
2.3. Equivalencia entre Autómatas Finitos Determinísticos y No Determinísticos . . .	12
2.4. Minimización de Autómatas Finitos Determinísticos	13
2.4.1. Indistinguibilidad	13
2.4.2. Autómata Finito Determinístico mínimo	16
2.4.3. Algoritmo de minimización	19
2.5. Equivalencia entre Autómatas Finitos y Gramáticas Regulares	22
2.6. Fuentes	25
3. Expresiones Regulares	26
3.1. Ecuaciones de Expresiones Regulares	26
3.2. Equivalencia entre Expresiones Regulares y Autómatas Finitos	26
3.3. Fuentes	31
4. Lenguajes Regulares	32
4.1. Lema de pumping para Lenguajes Regulares	32

4.2.	Unión de lenguajes regulares	34
4.3.	Intersección de lenguajes regulares	35
4.4.	Complemento de un lenguaje regular	36
4.5.	Lenguajes finitos	36
4.6.	Problemas decidibles acerca de lenguajes regulares	37
4.7.	Fuentes	37
5.	Autómatas de Pila	38
5.1.	Aceptación por estado final y por pila vacía	39
5.2.	Relación entre Autómatas de Pila y Gramáticas Libres de Contexto	43
5.3.	Autómatas de Pila Determinísticos	49
5.3.1.	Propiedad del prefijo	50
5.4.	Fuentes	50
6.	Lenguajes Libres de Contexto	51
6.1.	Árboles de derivación	52
6.2.	Ambigüedad	52
6.3.	Escritura y Reducción de Gramáticas Libres de Contexto	53
6.3.1.	Símbolos útiles	54
6.3.2.	Símbolos anulables	54
6.3.3.	Reglas de renombramiento	55
6.3.4.	Factorización a izquierda	56
6.3.5.	Recursividad a izquierda	56
6.3.6.	Ordenando las modificaciones	60
6.3.7.	Forma Normal de Chomsky	60
6.3.8.	Forma Normal de Greibach	60
6.4.	Lema de pumping para Lenguajes Libres de Contexto	60
6.5.	Propiedades de Lenguajes Libres de Contexto	62
6.5.1.	Unión de lenguajes libres de contexto	62
6.5.2.	Concatenación de lenguajes libres de contexto	62
6.5.3.	Clausura positiva de un lenguaje libre de contexto	63
6.5.4.	Clausura de Kleene de un lenguaje libre de contexto	63
6.5.5.	Intersección de lenguajes libres de contexto	63
6.6.	Lenguajes Libres de Contexto Determinísticos	64
6.7.	Lenguajes Libres de Contexto No Determinísticos	66
6.8.	Fuentes	67

7. Lenguajes y <i>Parsing LL</i>	68
7.1. Primeros de una cadena	69
7.2. Sigüientes de un símbolo no terminal	70
7.3. Símbolos Directrices	71
7.4. Ambigüedad	71
7.5. Recursividad a izquierda	72
7.6. Relación entre Lenguajes <i>LL</i> y Autómatas de Pila Determinísticos	72
7.7. Resultados sobre lenguajes y gramáticas <i>LL</i>	73
7.8. <i>Parsing LL(1)</i>	74
7.8.1. Tabla <i>LL(1)</i>	74
7.8.2. Algoritmo de <i>Parsing</i>	75
7.8.3. Complejidad del algoritmo de <i>parsing LL(1)</i>	77
7.9. Gramáticas extendidas <i>ELL</i>	78
7.10. Fuentes	79
8. Lenguajes y <i>Parsing LR</i>	80
8.1. Gramáticas <i>LR(k)</i>	80
8.2. Ambigüedad	81
8.3. Relación entre Lenguajes <i>LR</i> y Autómatas de Pila Determinísticos	82
8.4. Resultados sobre lenguajes y gramáticas <i>LR</i>	82
8.5. <i>Parsing LR</i>	84
8.5.1. Pivote	85
8.5.2. Prefijo viable	85
8.5.3. Ítems <i>LR(k)</i>	86
8.6. <i>Parsing LR</i>	88
8.6.1. Algoritmo de <i>Parsing LR(1)</i>	88
8.6.2. Complejidad del algoritmo de <i>parsing LR(k)</i>	90
8.6.3. Conflictos <i>LR</i> y variantes del algoritmo	92
8.7. Fuentes	92
9. <i>Parsing</i> generalizado para Lenguajes Libres de Contexto	93
9.1. <i>Parsing</i> de Cocke-Younger-Kasami	93
9.1.1. Algoritmo para pasar a forma normal de Chomsky	93
9.1.2. Algoritmo de construcción de la tabla CYK	94
9.1.3. Corrección de la tabla	96
9.1.4. Complejidad de la construcción de la tabla	97
9.1.5. Algoritmo de <i>parsing</i> CYK	97

9.1.6. Complejidad del Algoritmo CYK	98
9.2. <i>Parsing</i> de Earley	99
9.2.1. Ítem de Earley	99
9.2.2. Algoritmo de <i>parsing</i> de Earley	99
9.2.3. Corrección del algoritmo	100
9.2.4. Derivación a derecha de Earley	100
9.2.5. Complejidad del algoritmo	101
9.3. Fuentes	102
10. Gramáticas de Atributos	103
10.1. Atributos sintetizados y heredados	103
10.2. Agregando sensibilidad al contexto	104
10.3. Agregando semántica	105
10.3.1. Grafo de dependencias y gramática bien definida	105
10.3.2. Algoritmo de detección de circularidad	106
10.4. Gramáticas <i>s</i> -atribuidas	109
10.5. Gramáticas <i>l</i> -atribuidas	110
10.6. Gramáticas de atributos ordenadas	110
10.7. Traducción Dirigida por Sintaxis	110
10.8. Fuentes	111
11. Lenguajes Dependientes del Contexto y Lenguajes Recursivos	112
11.1. Máquinas de Turing	112
11.1.1. Transiciones de una Máquina de Turing	113
11.1.2. Equivalencia entre Máquinas de Turing Determinísticas y No Determinísticas	114
11.1.3. Codificación de Máquinas de Turing restringidas a un alfabeto binario	115
11.2. Autómatas Linealmente Acotados	115
11.3. Lenguajes Dependientes del Contexto	116
11.3.1. Relación entre Gramáticas Dependientes del Contexto y Autómatas Linealmente Acotados	117
11.3.2. Relación entre Lenguajes Dependientes del Contexto y Lenguajes Recursivos	117
11.4. Lenguajes Recursivos y Recursivamente Enumerables	118
11.4.1. Relación entre Gramáticas Sin Restricciones y Máquinas de Turing	119
11.4.2. Existencia de un lenguaje no recursivamente enumerable	121
11.4.3. Existencia de un lenguaje no recursivo	122
11.5. Fuentes	124

1. Lenguajes y Gramáticas

1.1. Repaso de relaciones

Definición 1.1. Dados dos conjuntos A y B , una **relación de A en B** es todo subconjunto de $A \times B$, es decir $R \subset A \times B$. Dos elementos $a \in A$, $b \in B$ están relacionados si $(a, b) \in R$. Notación: $a R b$.

Definición 1.2. Una relación $R : A \rightarrow A$ se dice **reflexiva** si todo elemento de A se relaciona consigo mismo. Es decir, cuando:

$$\forall a \in A, a R a$$

Definición 1.3. Una relación $R : A \rightarrow A$ se dice **simétrica** si cuando a se relaciona con b , también ocurre que b se relaciona con a . Es decir:

$$\forall a, b \in A : a R b \Rightarrow b R a$$

Definición 1.4. Una relación $R : A \rightarrow A$ se dice **transitiva** si vale que cuando a se relaciona con b y b con c , entonces también ocurre que a se relaciona con c . Es decir:

$$\forall a, b, c \in A : a R b \wedge b R c \Rightarrow a R c$$

Definición 1.5. Una relación $R : A \rightarrow A$ se dice **de equivalencia** cuando es reflexiva, simétrica y transitiva. Una relación de equivalencia R sobre A particiona al conjunto A en subconjuntos disjuntos llamados **clases de equivalencia**.

1.1.1. Composición

Definición 1.6. Sean A, B, C conjuntos, y sean las relaciones $R : A \rightarrow B$ y $G : B \rightarrow C$. Se define la relación de **composición** $G \circ R$ como:

$$G \circ R = \{(a, c), a \in A, c \in C : \exists b \in B \mid a R b \wedge b G c\}$$

Definición 1.7. Una relación $R : A \rightarrow A$ es de **identidad** (notado id_A) si se cumple que:

$$\forall a, b \in A : a id_A b \Leftrightarrow a = b$$

La relación de identidad es el elemento neutro de la composición.

Definición 1.8. (Potencia de relaciones) Dada una relación R sobre A , se define R^n como:

$$R^n = \begin{cases} id_A & \text{si } n = 0, \\ R \circ R^{n-1} & \text{si } n > 0 \end{cases}$$

1.1.2. Clausuras

Definición 1.9. Dada una relación R sobre A , se define su **clausura transitiva** R^+ como:

$$R^+ = \bigcup_{i=1}^{\infty} R^i = R \cup R^2 \cup R^3 \dots$$

Proposición 1.1. *La clausura transitiva de una relación R cumple:*

1. $R \subseteq R^+$
2. R^+ es transitiva
3. $\forall G : A \rightarrow A$ transitiva, $R \subseteq G \Rightarrow R^+ \subseteq G$

Definición 1.10. Dada una relación R sobre A , se define su **clausura reflexiva transitiva** R^* como:

$$R^* = \bigcup_{i=0}^{\infty} R^i = R^0 \cup R \cup R^2 \cup R^3 \dots = R^0 \cup R^+$$

Observación. Dada una relación $R : A \rightarrow A$ con A conjunto finito, R es finita. Esto es porque $R \subseteq A \times A$, y como A es finito, $A \times A$ también lo es.

Observación. Si R es una relación reflexiva, entonces $R^* = R^+$.

1.2. Lenguajes

Definición 1.11. Un **alfabeto** es un conjunto finito de elementos (caracteres).

Definición 1.12. Una **cadena** es un conjunto ordenado de elementos de un alfabeto.

Ejemplo 1.1. Para un alfabeto $\Sigma = \{a, b, c\}$, son cadenas posibles $aaabbbccc, abcc, cbbb, bcbbacb, \dots$

Concatenación: es una operación entre un símbolo de un alfabeto Σ y una cadena sobre dicho alfabeto. Por ejemplo, si $\Sigma = \{a, b, c\}$ y $\alpha = ab$ es la cadena, entonces $a \circ ab = aab$ es una cadena.

La cadena nula λ es el neutro de la concatenación:

$$\forall a \in \Sigma, a \circ \lambda = a$$

Definición 1.13. Un **lenguaje** sobre un alfabeto Σ es un conjunto (no necesariamente finito) de cadenas (finitas) sobre dicho alfabeto.

Definición 1.14. Sean L_1, L_2 lenguajes definidos sobre los alfabetos Σ_1 y Σ_2 respectivamente. Se define la **concatenación de lenguajes** L_1 y L_2 como el siguiente lenguaje sobre el alfabeto $\Sigma_1 \cup \Sigma_2$:

$$L_1 L_2 = \{xy : x \in L_1 \wedge y \in L_2\}$$

Esto da lugar a la definición de **potencia de lenguajes**:

$$L^n = \begin{cases} \{\lambda\} & \text{si } n = 0, \\ LL^{n-1} & \text{si } n > 0 \end{cases}$$

1.2.1. Clausuras

Definición 1.15. Dado un lenguaje L , se define la **clausura de Kleene** de L (notado L^*) como:

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

Intuitivamente, Σ^* son todas las cadenas de longitud 0 o más, formadas por elementos de Σ .

Definición 1.16. Dado un lenguaje L , se define la **clausura positiva** de L (notado L^+) como:

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

Intuitivamente, Σ^+ son todas las cadenas no nulas que se pueden formar con elementos de Σ .

Observación. De las definiciones de la clausura de Kleene y la clausura positiva se desprende que:

- $L^+ = LL^* = L^*L$
- $L^* = L^+ \cup \{\lambda\}$

1.3. Gramáticas

Definición 1.17. Una **gramática** es una 4-upla $G = \langle V_N, V_T, P, S \rangle$, donde:

- V_N es el conjunto de símbolos no terminales.
- V_T es el conjunto de símbolos terminales.
- P es el conjunto de producciones de la forma $\alpha \rightarrow \beta$.
- $S \in V_N$ es el símbolo inicial o distinguido.

Definición 1.18. Dada una producción $A \rightarrow \alpha$, se denomina a A como la **cabeza** de la producción, y a α como el **cuerpo** de la producción.

1.3.1. Derivaciones

Dada una gramática $G = \langle V_N, V_T, P, S \rangle$, una derivación es un proceso en el cual se parte del símbolo distinguido S , reemplazando recursivamente símbolos no terminales que coincidan con la cabeza de alguna producción en P , por su cuerpo.

Formalmente, se definen mediante la relación \Rightarrow_G : si $A \rightarrow \beta \in P$ es una producción de G , y $\alpha, \beta, \gamma \in (V_N \cup V_T)^*$, se dice que $\alpha A \gamma$ **deriva directamente** en $\alpha \beta \gamma$, y se nota:

$$\alpha A \gamma \Rightarrow_G \alpha \beta \gamma$$

Las potencias, y las clausuras transitiva y reflexiva transitiva se pueden aplicar a la relación de derivación \Rightarrow_G , siendo:

- $\xRightarrow[k]{G}$ representa una derivación en G de k pasos.
- $\xRightarrow{*}{G}$ representa una derivación en G de cero o más pasos.
- $\xRightarrow{+}{G}$ representa una derivación en G de uno o más pasos.

Definición 1.19. Una **forma sentencial** de una gramática G se define como:

- El símbolo inicial S es una forma sentencial de G .
- Si $\alpha\beta\gamma$ es una forma sentencial de G , y $(\beta \rightarrow \delta) \in P$, entonces $\alpha\delta\gamma$ es también una forma sentencial de G .

Las gramáticas sirven para generar lenguajes en base a su sintaxis:

Definición 1.20. El **lenguaje generado** por una gramática $G = \langle V_N, V_T, P, S \rangle$ se define como:

$$\mathcal{L}(G) = \{\alpha \in V_T^* : S \xRightarrow{+}_G \alpha\}$$

1.3.2. Árboles de derivación

En una derivación, cuando hay más de un símbolo no terminal para reemplazar, es necesario elegir cuál reemplazar primero, estableciéndose así un orden en la derivación. La idea de los árboles de derivación es representar las derivaciones abstrayendo el orden en el cual se aplicaron.

Definición 1.21. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática. Un **árbol de derivación** es aquel que cumple:

- Cada nodo posee una etiqueta, que pertenece al conjunto $V_N \cup V_T \cup \{\lambda\}$.
- La raíz tiene como etiqueta al símbolo distinguido S .
- Cada nodo interno (no hoja) tiene como etiqueta a un símbolo no terminal.
- Cada hoja tiene como etiqueta a un símbolo terminal, o λ .
- Si un nodo interno tiene etiqueta A y sus hijos tienen X_1, X_2, \dots, X_k , entonces $A \rightarrow X_1, A \rightarrow X_2, \dots, A \rightarrow X_k \in P$.
- Si una hoja está etiquetada como λ , entonces es el único hijo de su padre.

Definición 1.22. Sea $\mathcal{T}(A)$ un árbol de derivación para la cadena A , y sea X un nodo del árbol. Se define como **camino** de X a la cadena A, X_1, \dots, X_k, X tal que:

$$A \Rightarrow \dots X_1 \dots \Rightarrow \dots X_2 \dots \Rightarrow \dots \Rightarrow \dots X_k \dots \Rightarrow \dots X \dots$$

donde $A, X_1, \dots, X_k \in \mathcal{T}(A)$.

Definición 1.23. Sea $\mathcal{T}(A)$ un árbol de derivación para la cadena A . Se define como **longitud de un camino** de X a la cantidad de arcos del camino que une A con X en $\mathcal{T}(A)$.

Definición 1.24. Sea $\mathcal{T}(A)$ un árbol de derivación para la cadena A . Se define como **altura** del árbol $\mathcal{T}(A)$ a:

$$\max \{|\alpha X| : x \text{ es hoja de } \mathcal{T}(A), \text{ y } A\alpha x \text{ es un camino de } x\}$$

1.4. Clasificación de gramáticas

Las gramáticas se pueden clasificar de acuerdo a las restricciones que se imponga sobre sus producciones. Esto se conoce como la clasificación o jerarquía de Chomsky.

1.4.1. Gramáticas regulares (tipo 3)

Son las gramáticas más restrictivas de la clasificación de Chomsky; corresponden a aquellas gramáticas en las que las producciones tienen un símbolo no terminal a la izquierda (en la cabeza de la producción), y un símbolo terminal o un símbolo terminal y un no terminal a la derecha (en el cuerpo de la producción). Además, el símbolo no terminal, de existir en el cuerpo de la producción, debe aparecer siempre del mismo lado respecto al terminal.

Definición 1.25. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática. Se dice que G es **regular** si:

- todas las producciones de G son de la forma $A \rightarrow xB$ o $A \rightarrow x$, donde $A, B \in V_N$ y $x \in V_T^*$. En este caso, G se dice **lineal a derecha**. Una manera equivalente de expresar esto es decir que todas las producciones son de la forma $A \rightarrow aB$, o $A \rightarrow a$, o $A \rightarrow \lambda$.
- todas las producciones de G son de la forma $A \rightarrow Bx$ o $A \rightarrow x$, donde $A, B \in V_N$ y $x \in V_T^*$. En este caso, G se dice **lineal a izquierda**. Una manera equivalente de expresar esto es decir que todas las producciones son de la forma $A \rightarrow Ba$, o $A \rightarrow a$, o $A \rightarrow \lambda$.

Ejemplo 1.2. La gramática $G = \langle \{S, A, B, C\}, \{a, b, c\}, P, S \rangle$, con producciones $P = \{S \rightarrow aA, A \rightarrow aA, A \rightarrow bB, B \rightarrow bB, B \rightarrow bC, C \rightarrow cC, C \rightarrow c\}$, genera el lenguaje $L = \{a^n b^m c^k : n, m, k \geq 1\}$. G es regular.

Los lenguajes generados por estas gramáticas son los **lenguajes regulares**, que también pueden ser expresados mediante **expresiones regulares**, y reconocidos usando **autómatas finitos**.

1.4.2. Gramáticas libres de contexto (tipo 2)

El siguiente nivel en la clasificación de Chomsky son las gramáticas libres de contexto (o independientes del contexto). La restricción sobre sus producciones es que todas deben tener exactamente un símbolo no terminal en la cabeza, y una cadena cualquiera en el cuerpo.

Definición 1.26. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática. Se dice que G es **libre de contexto** si todas sus producciones son de la forma $A \rightarrow \alpha$, donde $A \in V_N$ y $\alpha \in (V_N \cup V_T)^*$.

De la definición se desprende que toda gramática regular es libre de contexto:

$$GR \subset GLC$$

Ejemplo 1.3. La gramática $G = \langle \{E, T, F\}, \{a, +, *, (,)\}, P, E \rangle$, con producciones $P = \{E \rightarrow E + T, E \rightarrow T, T \rightarrow T * F, T \rightarrow F, F \rightarrow (E), F \rightarrow a\}$ genera las expresiones aritméticas con suma y producto. Esta G es libre de contexto.

Los lenguajes generados por las GLC se llaman **lenguajes libres de contexto** (o independientes del contexto), y son los reconocidos por **autómatas de pila**.

1.4.3. Gramáticas dependientes del contexto (tipo 1)

Las gramáticas dependientes del contexto (o sensitivas, o sensibles, al contexto) relajan la restricción de las de tipo 2, ya que permiten que haya símbolos terminales en la cabeza de las producciones, mientras que la longitud de la cabeza sea menor que la del cuerpo.

Definición 1.27. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática. Se dice que G es **dependiente del contexto** si todas sus producciones son de la forma $\alpha \rightarrow \beta$, donde $\alpha, \beta \in (V_N \cup V_T)^*$ y $|\alpha| \leq |\beta|$.

Notar que esta restricción impide generar la cadena nula λ . Es por esto que la inclusión de las gramáticas de tipo 2 en las de tipo 1 no es completa: toda gramática libre de contexto que no tenga reglas de la forma $A \rightarrow \lambda$ (llamadas *reglas borradoras*, o producciones- λ) es una gramática dependiente del contexto.

Observación. En algunas definiciones, las gramáticas dependientes del contexto permiten que la regla $S \rightarrow \lambda$ pertenezca al conjunto de producciones; pero en ese caso, el símbolo S no puede aparecer en el cuerpo de ninguna regla.

Ejemplo 1.4. La gramática $G = \langle \{S, B, C\}, \{a, b, c\}, P, S \rangle$, con producciones $P = \{S \rightarrow aSBC, S \rightarrow abC, CB \rightarrow BC, bB \rightarrow bb, bC \rightarrow bc, cC \rightarrow cc\}$ genera el lenguaje $L = \{a^n b^n c^n : n \geq 1\}$. G es dependiente del contexto.

Los lenguajes generados por estas gramáticas se conocen como **lenguajes dependientes del contexto** (o sensibles al contexto), y corresponden a aquellos aceptados por **autómatas linealmente acotados**.

1.4.4. Gramáticas sin restricciones (tipo 0)

Como su nombre indica, estas gramáticas no imponen ninguna restricción sobre la forma de sus producciones. Este conjunto de gramáticas incluye a todas las gramáticas.

Estas gramáticas generan todos los lenguajes aceptados por una **máquina de Turing**; dichos lenguajes se denominan **lenguajes recursivamente enumerables** (r.e.). Un ejemplo de lenguaje r.e. es el conjunto de programas en \mathbb{C} que terminan.

1.5. Fuentes

- Julio Jacobo. Teoría de Lenguajes, Clase Teórica 1. Primer cuatrimestre, 2022.
- Verónica Becher. Teoría de Lenguajes, Clase Teórica 8. Primer cuatrimestre, 2022.

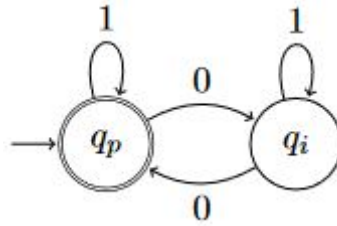
2. Autómatas Finitos

2.1. Autómatas Finitos Determinísticos

Definición 2.1. Un **autómata finito determinístico** (AFD) se define como una 5-upla $\langle Q, \Sigma, \delta, q_0, F \rangle$, donde:

- Q es el conjunto finito de estados.
- Σ es el alfabeto de entrada.
- $\delta : Q \times \Sigma \rightarrow Q$ es la función de transición.
- $q_0 \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados finales.

Ejemplo 2.1. El automáta $A = \langle \{q_p, q_i\}, \{0, 1\}, \delta, q_p, \{q_p\} \rangle$ reconoce las cadenas sobre $\Sigma = \{0, 1\}$ con un número par de ceros. A es un AFD, y su función de transición δ está dada por el siguiente diagrama:



La función de transición de un AFD puede extenderse para que acepte como segundo argumento cadenas en el alfabeto de entrada, y no sólo símbolos. Es decir, la versión extendida de la función de transición δ sería $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$, definida de la siguiente manera:

- $\hat{\delta}(q, \lambda) = q$
- $\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$, con $x \in \Sigma^*$ y $a \in \Sigma$

Intuitivamente, $\hat{\delta}(q, xa)$ se define como *hacer las transiciones para llegar desde el estado q hasta el estado resultante de consumir la cadena x , y luego hacer un paso más para consumir a .*

Observación. Como $\hat{\delta}(q, a) = \delta(\hat{\delta}(q, \lambda), a) = \delta(q, a)$, se puede usar el símbolo δ para referirse a ambos tipos de transición.

Definición 2.2. Se dice que una cadena x es **aceptada** por un AFD $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ cuando $\hat{\delta}(q_0, x) \in F$.

Esto da lugar a definir el lenguaje aceptado por un autómata finito determinístico, que estará compuesto por las cadenas aceptadas por el mismo:

Definición 2.3. Dado un AFD $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, el **lenguaje aceptado** por M se denota $\mathcal{L}(M)$, y se define como el conjunto de cadenas aceptadas por M ; es decir:

$$\mathcal{L}(M) = \{x : \hat{\delta}(q_0, x) \in F\}$$

2.2. Autómatas Finitos No Determinísticos

Definición 2.4. Un **autómata finito no determinístico** (AFND) se define como una 5-upla $\langle Q, \Sigma, \delta, q_0, F \rangle$, donde:

- Q es el conjunto finito de estados.
- Σ es el alfabeto de entrada.
- $\delta : Q \times \Sigma \rightarrow \mathcal{P}(Q)$ es la función de transición.
- $q_0 \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados finales.

Observación. La única diferencia en la 5-upla que define a un AFND respecto de un AFD es la función de transición, que en vez de retornar un estado, retorna un conjunto de estados.

Al igual que con los AFD, la función de transición δ se puede extender para aceptar como segundo argumento cadenas de Σ , es decir $\hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$. Esta versión extendida se define como:

- $\hat{\delta}(q, \lambda) = \{q\}$
- $\hat{\delta}(q, xa) = \{p : \exists r \in \hat{\delta}(q, x) \mid p \in \delta(r, a)\}$, donde $x \in \Sigma^*$ y $a \in \Sigma$

Si se generaliza la definición de δ para tomar conjuntos de estados, se tiene $\delta : \mathcal{P}(Q) \times \Sigma \rightarrow \mathcal{P}(Q)$ dada por:

$$\delta(P, a) = \bigcup_{q \in P} \delta(q, a)$$

Entonces, se puede escribir $\hat{\delta}(q, xa)$ como:

$$\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a)$$

Notar que esta última expresión es idéntica a la de la función de transición extendida para AFD.

Se puede extender la función de transición aún más, de manera que tome conjuntos de estados y cadenas, y retorne conjuntos de estados, es decir $\hat{\delta} : \mathcal{P}(Q) \times \Sigma^* \rightarrow \mathcal{P}(Q)$ definida como:

$$\hat{\delta}(P, x) = \bigcup_{q \in P} \hat{\delta}(q, x)$$

Se puede usar el mismo símbolo δ para aludir a todos los tipos de transición.

Definición 2.5. Se dice que una cadena x es **aceptada** por un AFND $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ cuando $\hat{\delta}(q_0, x) \cap F \neq \emptyset$. Es decir, cuando alguno de los estados resultantes de partir del estado inicial y consumir toda la cadena de entrada, es final.

Definición 2.6. Dado un AFND $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, el **lenguaje aceptado** por M se denota $\mathcal{L}(M)$, y se define como el conjunto de cadenas aceptadas por M ; es decir:

$$\mathcal{L}(M) = \{x : \hat{\delta}(q_0, x) \cap F \neq \emptyset\}$$

2.2.1. Autómatas Finitos no Determinísticos con transiciones λ

Los autómatas finitos no determinísticos con transiciones λ (AFND- λ) son iguales que los AFND comunes, con la distinción de que la función de transición recibe como segundo argumento no sólo algún símbolo del alfabeto de entrada, sino que también puede recibir *nada* (λ), produciéndose una *transición espontánea*.

Así, la definición de un AFND- λ es la misma que la de AFND (5-upla $\langle Q, \Sigma, \delta, q_0, F \rangle$), con los mismos componentes, salvo por la función de transición δ , cuya definición cambia de la siguiente manera, para poder aceptar λ como entrada:

$$\delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow \mathcal{P}(Q)$$

Definición 2.7. Se conoce como **clausura** λ de un estado q (notado $Cl_\lambda(q)$) al conjunto de estados alcanzables desde q , siguiendo sólo transiciones λ .

Observación. $q \in Cl_\lambda(q)$

Se puede generalizar la definición de clausura λ para conjuntos de estados P :

$$Cl_\lambda(P) = \bigcup_{q \in P} Cl_\lambda(q)$$

A su vez, la función de transición se puede extender para que acepte, como segundo argumento, cadenas en el alfabeto, y aplicando la clausura λ al resultado de la transición. Así, la función $\hat{\delta} : Q \times \Sigma^* \rightarrow \mathcal{P}(Q)$ se define como:

- $\hat{\delta}(q, \lambda) = Cl_\lambda(q)$
- $\hat{\delta}(q, xa) = Cl_\lambda(\{p : \exists r \in \hat{\delta}(q, x) \mid p \in \delta(r, a)\})$

donde $x \in \Sigma^*$, y $a \in \Sigma$. Es decir:

$$\hat{\delta}(q, xa) = Cl_\lambda\left(\bigcup_{r \in \hat{\delta}(q, x)} \delta(r, a)\right)$$

Al igual que con los AFND, se puede extender las funciones δ y $\hat{\delta}$ para conjuntos de estados:

- $\delta(P, a) = \bigcup_{q \in P} \delta(q, a)$
- $\hat{\delta}(P, x) = \bigcup_{q \in P} \hat{\delta}(q, x)$

Usando esto, se puede reescribir $\hat{\delta}(q, xa)$ como:

$$\hat{\delta}(q, xa) = Cl_\lambda(\delta(\hat{\delta}(q, x), a))$$

Observación. $\hat{\delta}(q, a)$ puede ser distinto de $\delta(q, a)$:

$$\hat{\delta}(q, a) = Cl_\lambda(\delta(\hat{\delta}(q, \lambda), a)) = Cl_\lambda(\delta(Cl_\lambda(q), a)) \neq \delta(q, a)$$

Por eso, para el caso de los AFND- λ , será necesario mantener la diferencia entre δ y $\hat{\delta}$.

Definición 2.8. Se dice que una cadena x es **aceptada** por un AFND- λ $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ cuando $\hat{\delta}(q_0, x) \cap F \neq \emptyset$.

Definición 2.9. Dado un AFND- λ $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, el **lenguaje aceptado** por M se denota $\mathcal{L}(M)$, y se define como el conjunto de cadenas aceptadas por M ; es decir:

$$\mathcal{L}(M) = \{x : \hat{\delta}(q_0, x) \cap F \neq \emptyset\}$$

Observación. La definición de cadena aceptada (y de lenguaje aceptado) para AFND- λ es la misma que para AFND.

Teorema 2.1. (*Equivalencia entre AFND y AFND- λ*) Dado un AFND- λ $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, existe un AFND $M' = \langle Q, \Sigma, \delta', q_0, F' \rangle$ tal que $\mathcal{L}(M) = \mathcal{L}(M')$.

Demostración. Tomando la siguiente definición para el conjunto F' de estados finales del autómata M' :

$$F' = \begin{cases} F \cup \{q_0\} & \text{si } Cl_\lambda(q_0) \cap F \neq \emptyset, \\ F & \text{en caso contrario} \end{cases}$$

Para la función de transición de M' , tomar $\delta'(q, a) = \hat{\delta}(q, a)$.

Como paso intermedio, se probará por inducción en $|x|$ que $\delta'(q_0, x) = \hat{\delta}(q_0, x)$ para las cadenas x tales que $|x| \geq 1$:

- Caso base: $|x| = 1$, es decir $x = a$. Por lo tanto, por definición de δ' se tiene que $\delta'(q, x) = \delta'(q, a) = \hat{\delta}(q, a) = \hat{\delta}(q, x)$.
- Paso inductivo: suponiendo que vale la proposición para toda cadena w de largo $|w|$. Considerar entonces la cadena $x = wa$ (con largo $|w| + 1$):

$$\begin{aligned} \delta'(q_0, wa) &= \delta'(\delta'(q_0, w), a), \text{ por definición de función de transición.} \\ &= \delta'(\hat{\delta}(q_0, w), a), \text{ por hipótesis inductiva.} \end{aligned}$$

Por otro lado, es inmediato que para $P \subseteq Q$ es cierto que $\delta'(P, a) = \hat{\delta}(P, a)$, pues:

$$\delta'(P, a) = \bigcup_{q \in P} \delta'(q, a) = \bigcup_{q \in P} \hat{\delta}(q, a) = \hat{\delta}(P, a)$$

Luego, tomando $P = \hat{\delta}(q_0, w)$, se tiene que:

$$\delta'(q_0, wa) = \delta'(\hat{\delta}(q_0, w), a) = \hat{\delta}(\hat{\delta}(q_0, w), a) = \hat{\delta}(q_0, wa)$$

que es lo que se quería probar.

Con eso concluye la inducción, quedando probado que $\delta'(q_0, x) = \hat{\delta}(q_0, x)$ para las cadenas x tales que $|x| \geq 1$.

Falta ahora demostrar que $\mathcal{L}(M) = \mathcal{L}(M')$; es decir, que para toda cadena x , $x \in \mathcal{L}(M) \Leftrightarrow x \in \mathcal{L}(M')$. Para eso se separará en dos casos:

- Caso 1: $x = \lambda$.

$$\lambda \in \mathcal{L}(M)$$

$\Leftrightarrow \hat{\delta}(q_0, \lambda) \cap F \neq \emptyset$, por definición de cadena aceptada por AFND- λ .

$\Leftrightarrow Cl_\lambda(q_0) \cap F \neq \emptyset$, por definición de $\hat{\delta}(q, \lambda)$ en AFND- λ .

$\Rightarrow q_0 \in F'$, por definición de F' .

$\Leftrightarrow \lambda \in \mathcal{L}(M')$, porque si q_0 es estado final, λ es aceptada.

Ahora la recíproca:

$$\lambda \in \mathcal{L}(M')$$

$\Leftrightarrow q_0 \in F'$, porque si λ es aceptada, q_0 es estado final.

$\Rightarrow q_0 \in F \vee Cl_\lambda(q_0) \cap F \neq \emptyset$, por definición de F' .

$\Rightarrow Cl_\lambda(q_0) \cap F \neq \emptyset \vee \lambda \in \mathcal{L}(M)$, por definición de F y de cadena aceptada en AFND- λ .

$\Rightarrow \lambda \in \mathcal{L}(M) \vee \lambda \in \mathcal{L}(M)$, por definición de cadena aceptada en AFND- λ .

$\Rightarrow \lambda \in \mathcal{L}(M)$

Luego, vale que:

$$\lambda \in \mathcal{L}(M) \Leftrightarrow \lambda \in \mathcal{L}(M')$$

- Caso 2: $x \neq \lambda$.

$$x \in \mathcal{L}(M)$$

$\Leftrightarrow \hat{\delta}(q_0, x) \cap F \neq \emptyset$, por definición de cadena aceptada en AFND- λ .

$\Rightarrow \delta'(q_0, x) \cap F' \neq \emptyset$, por el resultado intermedio, y por $F \subset F'$.

$\Rightarrow x \in \mathcal{L}(M')$, por definición de cadena aceptada en AFND.

Ahora la recíproca:

$$x \in \mathcal{L}(M')$$

$\Leftrightarrow \delta'(q_0, x) \cap F' \neq \emptyset$, por definición de cadena aceptada en AFND.

$\Rightarrow \hat{\delta}(q_0, x) \cap F \neq \emptyset$

$\vee \left(\hat{\delta}(q_0, x) \cap \{q_0\} \neq \emptyset \wedge Cl_\lambda(q_0) \cap F \neq \emptyset \right)$, por el resultado intermedio y por definición de F' .

$\Rightarrow x \in \mathcal{L}(M)$

\vee (existe un *loop* x de q_0 sobre sí mismo \wedge

existe un camino λ desde q_0 hasta F , por lo que existe un camino x desde q_0 hasta F)

$\Rightarrow x \in \mathcal{L}(M) \vee x \in \mathcal{L}(M)$

$\Rightarrow x \in \mathcal{L}(M)$

Luego, vale que para toda cadena x :

$$x \in \mathcal{L}(M) \Leftrightarrow x \in \mathcal{L}(M')$$

□

Observación. La conclusión principal de este teorema es que los AFND y los AFND- λ tienen el mismo poder expresivo.

2.3. Equivalencia entre Autómatas Finitos Determinísticos y No Determinísticos

Trivialmente se puede afirmar que **para todo AFD, existe un AFND equivalente**. Sin embargo, la afirmación recíproca no es trivial: que **para cada AFND existe un AFD equivalente**:

Teorema 2.2. *Dado un AFND $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, existe un AFD $M' = \langle Q', \Sigma, \delta', q'_0, F' \rangle$ tal que $\mathcal{L}(M) = \mathcal{L}(M')$.*

Demostración. Se construye un AFD M' , cuyos estados se denotan $[q_1, \dots, q_i]$, con $q_1, \dots, q_i \in Q$ y corresponden a $\mathcal{P}(Q)$ (es decir, los estados de M' son conjuntos de estados de M). El conjunto de estados finales en M' será

$$F' = \{[q_1, \dots, q_i] \in Q' : \{q_1, \dots, q_i\} \cap F \neq \emptyset\}$$

Y el estado inicial será $q'_0 = [q_0]$. La función de transición δ' se define como:

$$\delta'([q_1, \dots, q_j], a) = [p_1, \dots, p_i] \Leftrightarrow \delta(\{q_1, \dots, q_j\}, a) = \{p_1, \dots, p_i\}$$

Como paso intermedio, se probará por inducción en la longitud de la cadena x , que: para toda cadena x vale

$$\delta'(q'_0, x) = [q_1, \dots, q_i] \Leftrightarrow \delta(q_0, x) = \{q_1, \dots, q_i\}$$

- Caso base: $|x| = 0$, i.e. $x = \lambda$: debido a la definición de la función de transición δ generalizada, se tiene:

$$\begin{aligned} \delta'(q'_0, \lambda) &= [q_0] \quad , \text{ y} \\ \delta(q_0, \lambda) &= \{q_0\} \end{aligned}$$

Por lo tanto,

$$\delta'(q'_0, \lambda) = [q_0] \Leftrightarrow \delta(q_0, \lambda) = \{q_0\}$$

- Paso inductivo: suponiendo que vale la proposición para las cadenas x , considerar el caso de la cadena xa , de largo $|x| + 1$:

$\delta'(q'_0, xa) = \delta'(\delta'(q'_0, x), a)$, por definición de la función de transición en AFDs.

$\delta(q_0, xa) = \delta(\delta(q_0, x), a)$, por definición de la función de transición en AFNDs.

$\delta'(q'_0, x) = [p_1, \dots, p_k] \Leftrightarrow \delta(q_0, x) = \{p_1, \dots, p_k\}$, por hipótesis inductiva.

$\delta'([p_1, \dots, p_k], a) = [r_1, \dots, r_i] \Leftrightarrow \delta(\{p_1, \dots, p_k\}, a) = \{r_1, \dots, r_i\}$, por definición de δ' .

Luego,

$$\delta'(q'_0, xa) = \delta'(\delta'(q'_0, x), a) = [r_1, \dots, r_i]$$

$$\Leftrightarrow \exists [p_1, \dots, p_k] \mid \delta'(q'_0, x) = [p_1, \dots, p_k] \wedge \delta'([p_1, \dots, p_k], a) = [r_1, \dots, r_i] \text{ , por definición de } \delta \text{ en AFDs.}$$

$$\Leftrightarrow \exists \{p_1, \dots, p_k\} \mid \delta(q_0, x) = \{p_1, \dots, p_k\} \wedge \delta(\{p_1, \dots, p_k\}, a) = \{r_1, \dots, r_i\} \text{ , por hipótesis inductiva.}$$

$$\Leftrightarrow \delta(q_0, xa) = \delta(\delta(q_0, x), a) = \{r_1, \dots, r_i\} \text{ , por definición de } \delta \text{ en AFNDs.}$$

Por lo tanto,

$$\delta'(q'_0, xa) = [r_1, \dots, r_i] \Leftrightarrow \delta(q_0, xa) = \{r_1, \dots, r_i\}$$

Con lo cual concluye la inducción, quedando así probado para toda cadena x , que:

$$\delta'(q'_0, x) = [q_1, \dots, q_i] \Leftrightarrow \delta(q_0, x) = \{q_1, \dots, q_i\}$$

Ahora queda por probar que $\mathcal{L}(M) = \mathcal{L}(M')$:

$$x \in \mathcal{L}(M)$$

$$\Leftrightarrow \delta(q_0, x) = \{q_1, \dots, q_i\}$$

$$\wedge \{q_1, \dots, q_i\} \cap F \neq \emptyset, \text{ por definición de lenguaje definido por AFNDs.}$$

$$\Leftrightarrow \delta'(q'_0, x) = [q_1, \dots, q_i]$$

$$\wedge [q_1, \dots, q_i] \in F', \text{ por lo que se acaba de probar por inducción, y por definición de } F'.$$

$$\Leftrightarrow x \in \mathcal{L}(M'), \text{ por definición de lenguaje definido por AFDs.}$$

□

Observación. La conclusión principal de este teorema es que los AFD y los AFND (y por lo tanto también los AFND- λ , por el Teorema 2.1) tienen el mismo poder expresivo.

La idea de esta construcción es la siguiente: el lenguaje aceptado por un AFND son todas las cadenas para las cuales existe un camino (en el diagrama de la función de transición) a un estado final. Entonces, para lograr introducir el determinismo, la idea será mirar todos los posibles caminos a la vez (por eso los estados son conjuntos de estados). Entonces, en cada paso se está en un conjunto de estados todos a la vez, y cada transición lleva a otro conjunto de estados. Consumir una cadena lleva determinísticamente a un conjunto de estados, que serán todos aquellos a los que se podía llegar de manera no determinística en el AFND original. Es por eso que sólo se aceptan cadenas que lleven a conjuntos de estados en los que haya alguno final.

Observación. La cantidad de estados del AFD resultante de la construcción descrita es potencialmente mucho mayor que la del AFND original. De hecho, podría tener exponencialmente mayor cantidad de estados (es la cardinalidad del conjunto de partes).

2.4. Minimización de Autómatas Finitos Determinísticos

Para paliar el aumento exponencial de estados que se produce al transformar un AFND en un AFD, se utilizan ciertos mecanismos bajo el nombre de *minimización* de AFDs. La idea del método que se estudia en la materia es eliminar los estados del autómata que *no aporten información extra*; esto es, si hay dos estados a partir de los cuales *se puede generar el mismo lenguaje* (i.e. aceptar las mismas cadenas), se elimina alguno de ellos. Para eso, se introduce el concepto de indistinguibilidad.

2.4.1. Indistinguibilidad

Definición 2.10. Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un autómata finito determinístico (AFD). Se dice que los estados $p, q \in Q$ son **indistinguibles** cuando: para toda cadena x , partiendo del estado p se llega a un estado final, si, y sólo si, partiendo desde el estado q también se llega a un estado final. Es decir:

$$p \equiv q \stackrel{\Delta}{\Leftrightarrow} \forall x \in \Sigma^* : (\hat{\delta}(p, x) \in F \Leftrightarrow \hat{\delta}(q, x) \in F)$$

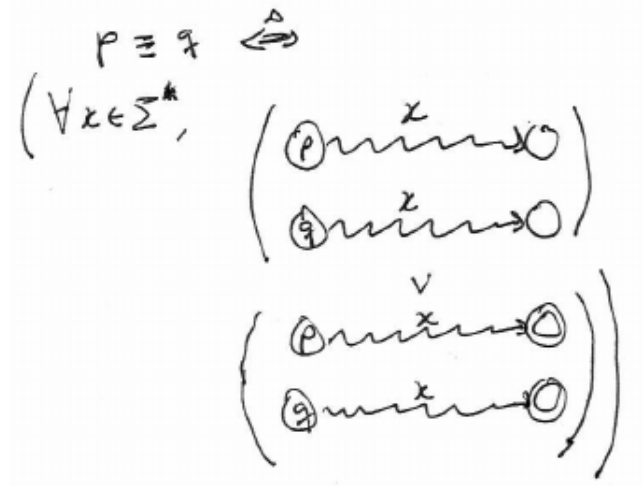


Figura 1: Interpretación gráfica de la noción de indistinguibilidad.

Teorema 2.3. Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un autómata finito determinístico (AFD), y sea $p, q \in Q$ un par de estados indistinguibles, y $x \in \Sigma^*$ una cadena cualquiera. Entonces, al M consumir x , el par de estados resultantes será también indistinguible (ver Figura 2 para una interpretación gráfica). Es decir:

$$p \equiv q \Rightarrow \forall x \in \Sigma^* : (\hat{\delta}(p, x) \equiv \hat{\delta}(q, x))$$

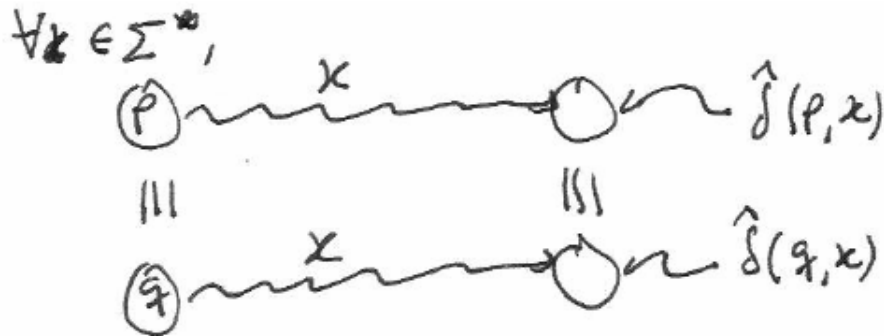
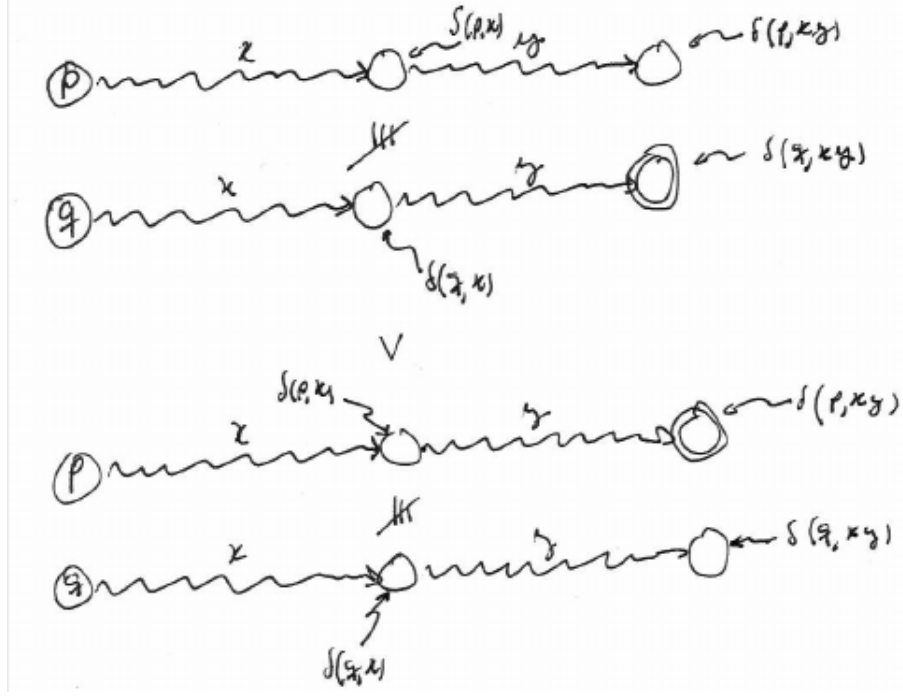


Figura 2: Interpretación gráfica del Teorema 2.3.

Demostración. Procediendo por el absurdo: suponer que

$$\exists x \in \Sigma^* \mid (\hat{\delta}(p, x) \not\equiv \hat{\delta}(q, x))$$

Es decir, ocurre alguno de los dos escenarios siguientes:



Luego, existe una cadena y que distingue a $\hat{\delta}(p, x)$ de $\hat{\delta}(q, x)$ (o viceversa), es decir, según la definición de indistinguibilidad:

$$\exists y \in \Sigma^* \mid \left(\hat{\delta}(\hat{\delta}(p, x), y) \in F \wedge \hat{\delta}(\hat{\delta}(q, x), y) \notin F \right)$$

, o viceversa. Esto equivale a afirmar:

$$\hat{\delta}(p, xy) \in F \wedge \hat{\delta}(q, xy) \notin F$$

o viceversa. Pero entonces, p lleva a un estado final al consumir la cadena xy , mientras que q lleva a un estado no final al consumir la misma cadena (o viceversa). Con lo cual, por definición de indistinguibilidad, $p \not\equiv q$. Esto es absurdo, ya que por hipótesis se tenía que $p \equiv q$. La contradicción proviene de haber supuesto lo contrario a lo que se quería probar. Por lo tanto, queda demostrado el resultado. \square

Teorema 2.4. *La indistinguibilidad \equiv es una relación de equivalencia.*

Demostración. Probando que \equiv es reflexiva, simétrica y transitiva:

- Reflexividad: por definición,

$$\forall \alpha \in \Sigma^* : \left(\hat{\delta}(q, \alpha) \in F \Leftrightarrow \hat{\delta}(q, \alpha) \in F \right) \Rightarrow q \equiv q$$

- Simetría: por definición, si $q \equiv r$ entonces,

$$\forall \alpha \in \Sigma^* : \left(\hat{\delta}(q, \alpha) \in F \Leftrightarrow \hat{\delta}(r, \alpha) \in F \right)$$

Esto es equivalente a la siguiente expresión:

$$\forall \alpha \in \Sigma^* : \left(\hat{\delta}(r, \alpha) \in F \Leftrightarrow \hat{\delta}(q, \alpha) \in F \right)$$

Por definición, esto implica que $r \equiv q$.

- Transitividad: partiendo de que $q \equiv r$ y $r \equiv s$, entonces por definición se tiene:

$$\left(\forall \alpha \in \Sigma^* : \left(\hat{\delta}(q, \alpha) \in F \Leftrightarrow \hat{\delta}(r, \alpha) \in F \right) \right) \wedge \left(\forall \alpha \in \Sigma^* : \left(\hat{\delta}(r, \alpha) \in F \Leftrightarrow \hat{\delta}(s, \alpha) \in F \right) \right)$$

De las expresiones anteriores, surge:

$$\forall \alpha \in \Sigma^* : \left(\hat{\delta}(q, \alpha) \in F \Leftrightarrow \hat{\delta}(s, \alpha) \in F \right)$$

Por definición, esto implica que $q \equiv s$.

□

2.4.2. Autómata Finito Determinístico mínimo

Ya introducido el concepto de indistinguibilidad y algunos resultados sobre el mismo, se introduce ahora la noción de un AFD mínimo:

Definición 2.11. Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un autómata finito determinístico (AFD) sin estados inaccesibles. El **AFD mínimo equivalente** a M será el AFD $M' = \langle Q', \Sigma', \delta', q'_0, F' \rangle$, dado por:

- $Q' = (Q / \equiv)$ (las clases de equivalencia introducidas por la relación de equivalencia \equiv en el conjunto de estados Q)
- $\delta'([q], a) = [\delta(q, a)]$ ($[q]$ es la clase de equivalencia del estado q)
- $q'_0 = [q_0]$
- $F' = \{[q] \in Q' : q \in F\}$

Ahora, se verá que esta construcción preserva la definición de la función de transición:

Teorema 2.5. Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un autómata finito determinístico (AFD) sin estados inaccesibles, y sea $M' = \langle Q', \Sigma', \delta', q'_0, F' \rangle$ su AFD mínimo equivalente, definido como en la Definición 2.11. Entonces:

$$\forall x \in \Sigma^* : \hat{\delta}(q, x) = r \Rightarrow \hat{\delta}'([q], x) = [r]$$

Demostración. Por inducción en $|x|$:

- Caso base: $|x| = 0$, i.e. $x = \lambda$.
Por definición de $\hat{\delta}$, vale que $\hat{\delta}(q, \lambda) = q$.
Por definición de $\hat{\delta}'$, vale que $\hat{\delta}'([q], \lambda) = [q]$.
De las dos afirmaciones anteriores se deduce que:

$$\hat{\delta}(q, \lambda) = q \Rightarrow \hat{\delta}'([q], \lambda) = [q]$$

- **Paso inductivo:** suponiendo que la proposición vale para las cadenas de largo $n \geq 0$, se busca probarla para cadenas x tales que $|x| = n + 1$.

Tomando entonces la cadena xa con $|x| = n$, hay que probar que $\hat{\delta}(q, xa) = r \Rightarrow \hat{\delta}'([q], xa) = [r]$.

Por hipótesis inductiva, vale que $\hat{\delta}(q, x) = p \Rightarrow \hat{\delta}'([q], x) = [p]$. Entonces:

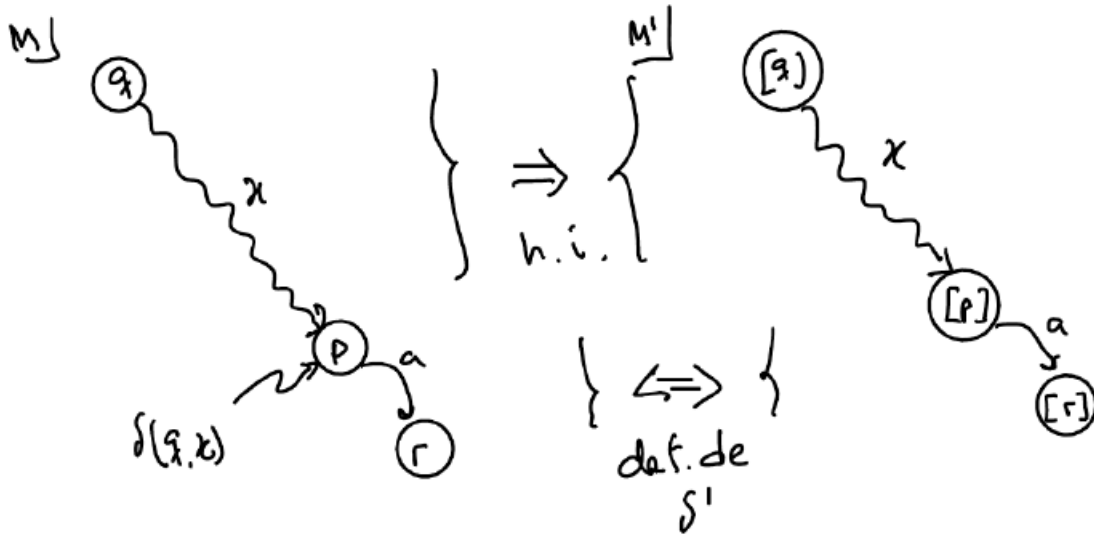
$$\hat{\delta}(q, xa) = \delta(\hat{\delta}(q, x), a) = \delta(p, a) = r \Rightarrow \delta'([p], a) = [r]$$

, donde la **implicación** proviene de la definición de δ' .

Aplicando la hipótesis inductiva, se tiene que $\hat{\delta}'([q], x) = [p]$, por lo que reemplazando $[p]$ en el consecuente de esta implicación, se obtiene:

$$\delta'([p], a) = \delta'(\hat{\delta}'([q], x), a) = \hat{\delta}'([q], xa) = [r]$$

En la siguiente imagen se puede ver gráficamente la idea de este razonamiento:



□

Para dar con los fundamentos que permitirán definir un algoritmo para minimizar un AFD, se verá ahora el concepto de **indistinguibilidad de orden k** .

Definición 2.12. Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un autómata finito determinístico (AFD). Se dice que los estados $p, q \in Q$ son **indistinguibles de orden k** cuando p y q son indistinguibles, pero considerando cadenas de longitud menor o igual que k . Es decir:

$$p \stackrel{k}{\equiv} q \triangleq \forall x \in \Sigma^* : (|x| \leq k) \Rightarrow (\hat{\delta}(p, x) \in F \Leftrightarrow \hat{\delta}(q, x) \in F)$$

Intuitivamente, $p \stackrel{k}{\equiv} q$ si, mirando el lenguaje generado a partir de p considerando sólo las palabras de longitud hasta k , y mirando el lenguaje generado a partir de q considerando sólo las palabras de longitud hasta k , estos dos conjuntos coinciden.

La relación de indistinguibilidad de orden k cumple una serie de propiedades, que permiten dar con el algoritmo de minimización que se estudia en la materia.

Teorema 2.6. Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un autómata finito determinístico (AFD). La relación de indistinguibilidad de orden k (\equiv^k) cumple las siguientes propiedades:

1. \equiv^k es una relación de equivalencia.
2. $\equiv^{k+1} \subseteq \equiv^k$
3. $\left(Q / \equiv^0 \right) = \{ Q \setminus F, F \}$ si $Q \setminus F \neq \emptyset$ y $F \neq \emptyset$
4. $p \equiv^{k+1} q \Leftrightarrow \left(p \equiv^0 q \right) \wedge \left(\forall a \in \Sigma : \delta(p, a) \equiv^k \delta(q, a) \right)$
5. $\left(\equiv^{k+1} = \equiv^k \right) \Rightarrow \forall n \geq 0 : \left(\equiv^{k+n} = \equiv^k \right)$

Demostración. Yendo por partes:

1. La demostración es análoga a la del Teorema 2.4.
2. Esta proposición dice que la $(k+1)$ -indistinguibilidad implica la k -indistinguibilidad. Partiendo de la hipótesis $p \equiv^{k+1} q$, se tiene que por definición:

$$\forall x \in \Sigma^* : (|x| \leq k+1) \Rightarrow (\hat{\delta}(p, x) \in F \Leftrightarrow \hat{\delta}(q, x) \in F) \quad (1)$$

Por otro lado, es cierto que:

$$(|x| \leq k) \Rightarrow (|x| \leq k+1) \quad (2)$$

Luego, juntando (1) y (2), se deduce que:

$$\forall x \in \Sigma^* : (|x| \leq k) \Rightarrow (\hat{\delta}(p, x) \in F \Leftrightarrow \hat{\delta}(q, x) \in F)$$

Y eso es equivalente, por definición, a afirmar que $p \equiv^k q$.

3. Surge de la definición de \equiv^k con $k = 0$.

4. Viendo las dos implicaciones por separado:

- \Rightarrow) Se probó en 2 que $\equiv^{k+1} \subseteq \equiv^k$, lo cual implica que: $p \equiv^{k+1} q \Rightarrow p \equiv^0 q$. Entonces, falta ver que $\left(\forall a \in \Sigma : \delta(p, a) \equiv^k \delta(q, a) \right)$. Suponiendo que esto no es cierto, entonces vale que:

$$\exists a \in \Sigma, \exists \alpha \in \Sigma^* \mid (|\alpha| \leq k) \wedge \left(\hat{\delta}(\delta(p, a), \alpha) \in F \right) \wedge \left(\hat{\delta}(\delta(q, a), \alpha) \notin F \right) \quad (3)$$

o viceversa. Pero entonces $p \not\equiv^{k+1} q$, pues $|a\alpha| \leq k+1$, y $a\alpha$ distingue p de q . Esto contradice la hipótesis, y la contradicción proviene de haber supuesto (3). Por lo tanto, vale \Rightarrow).

- \Leftarrow) Suponer que $p \not\equiv^{k+1} q$ (es decir, que no vale la implicación). Entonces, vale que o $p \not\equiv^0 q$, o bien que hay una cadena de longitud menor o igual que $k+1$ que distingue p de q , es decir:

$$\exists a \in \Sigma, \exists \alpha \in \Sigma^* \mid (|\alpha| \leq k) \wedge \left(\hat{\delta}(\delta(p, a), \alpha) \in F \right) \wedge \left(\hat{\delta}(\delta(q, a), \alpha) \notin F \right)$$

o viceversa. Pero entonces $\delta(p, a) \stackrel{k}{\not\equiv} \delta(q, a)$ por definición de indistinguibilidad de orden k . Esto contradice parte de la hipótesis, que decía que $\left(\forall a \in \Sigma : \delta(p, a) \stackrel{k}{\equiv} \delta(q, a)\right)$.

Por otro lado, por lo que se supuso, si no vale esto entonces vale que $p \stackrel{0}{\not\equiv} q$, lo cual contradice la otra parte de la hipótesis, que decía que $p \stackrel{0}{\equiv} q$. De cualquier forma se llega a una contradicción, proveniente de haber supuesto que no valía la implicación \Leftarrow ; por lo tanto, sí vale.

5. Procediendo por inducción en $n \geq 0$:

- Caso base: $n = 0$. Vale *trivialmente*, pues reemplazando $n = 0$ se obtiene que $\left(\stackrel{k}{\equiv} = \stackrel{k}{\equiv}\right)$.
- Paso inductivo: suponiendo que vale la proposición para n , es decir:

$$\left(\stackrel{k+1}{\equiv} = \stackrel{k}{\equiv}\right) \Rightarrow \left(\stackrel{k+n}{\equiv} = \stackrel{k}{\equiv}\right)$$

, hay que probar que vale para $n + 1$, es decir:

$$\left(\stackrel{k+1}{\equiv} = \stackrel{k}{\equiv}\right) \Rightarrow \left(\stackrel{k+n+1}{\equiv} = \stackrel{k}{\equiv}\right) \quad (4)$$

Por definición de igualdad de relaciones, es cierto que:

$$\left(\stackrel{k+n+1}{\equiv} = \stackrel{k}{\equiv}\right) \Leftrightarrow \forall p, q \in Q : \left(q \stackrel{k}{\equiv} p \Leftrightarrow q \stackrel{k+n+1}{\equiv} p\right)$$

Por lo tanto, (4) se puede reescribir como:

$$\left(\stackrel{k+1}{\equiv} = \stackrel{k}{\equiv}\right) \Rightarrow \forall p, q \in Q : \left(q \stackrel{k}{\equiv} p \Leftrightarrow q \stackrel{k+n+1}{\equiv} p\right) \quad (5)$$

Así, demostrar (5) es lo mismo que demostrar (4), que es lo que hay que probar. Esto se puede demostrar de la siguiente manera:

$$\begin{aligned} q \stackrel{k+n+1}{\equiv} p &\Leftrightarrow \left(q \stackrel{0}{\equiv} p\right) \wedge \left(\forall a \in \Sigma : \delta(q, a) \stackrel{k+n}{\equiv} \delta(p, a)\right), \text{ por 4} \\ &\Leftrightarrow \left(q \stackrel{0}{\equiv} p\right) \wedge \left(\forall a \in \Sigma : \delta(q, a) \stackrel{k}{\equiv} \delta(p, a)\right), \text{ por la hipótesis inductiva,} \\ &\quad \text{y porque vale el antecedente de (5), lo cual garantiza el consecuente} \\ &\Leftrightarrow q \stackrel{k+1}{\equiv} p, \text{ por 4} \\ &\Leftrightarrow q \stackrel{k}{\equiv} p, \text{ porque vale el antecedente de (5)} \end{aligned}$$

Con eso queda probado el resultado para todo $n \geq 0$.

□

2.4.3. Algoritmo de minimización

La idea es que con las propiedades del Teorema 2.6 se puede construir un algoritmo que, dado un AFD sin estados inaccesibles, produzca un AFD equivalente que sea mínimo, para esto, el AFD a construir tendrá todos sus estados indistinguibles:

- Al ser \equiv^k una relación de equivalencia (por 1), define una partición en el conjunto de estados Q del autómata, para cada k .
- Por 2, vale que se puede ir haciendo un *refinamiento* progresivo de la partición.
- Por 3, se conoce por dónde empezar la construcción.
- Por 4, hay una manera de pasar de la indistinguibilidad de orden k a la de orden $k + 1$.
- Por 5, se sabe que el algoritmo va a terminar.

Ahora se muestra pseudocódigo para el algoritmo de minimización:

Algoritmo 1: *Minimización de un AFD*

```

1:  $P \leftarrow \{Q \setminus F, F\}$                                 ▷ con los elementos de  $Q \setminus F$  y  $F$  sin marcar
2: repeat
3:    $P' \leftarrow \emptyset$ 
4:   for each  $X \in P$  do                                ▷ particionar cada  $X \in P$ 
5:     while  $(\exists e \in X) \wedge (\neg \text{marked}(e, X))$  do
6:        $X_1 \leftarrow \{e\}$                                 ▷ con  $e$  sin marcar
7:        $\text{mark}(e, X)$                                        ▷ marcar  $e$  en  $X$ 
8:       for each  $e' \in X$  do
9:         if  $\neg \text{marked}(e', X) \wedge (\forall a \in \Sigma : [\delta(e, a)] = [\delta(e', a)])$  then
10:           $X_1 \leftarrow X_1 \cup \{e'\}$ 
11:           $\text{mark}(e', X)$                                    ▷ marcar  $e'$  en  $X$ 
12:        end if
13:      end for
14:       $P' \leftarrow P' \cup \{X_1\}$ 
15:    end while
16:  end for
17:  if  $P' \neq P$  then                                     ▷ ¿las particiones actual y anterior son distintas?
18:     $\text{stop} \leftarrow \text{False}$                                 ▷ si son distintas, seguir
19:     $P \leftarrow P'$ 
20:  else
21:     $\text{stop} \leftarrow \text{True}$                                    ▷ si son iguales, parar
22:  end if
23: until  $\text{stop} = \text{True}$ 

```

Para demostrar la corrección de este algoritmo, se presentan los siguientes resultados:

Lema 2.1. Sean $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, y $M' = \langle Q', \Sigma, \delta', q'_0, F \rangle$ dos AFDs, donde M no posee estados inaccesibles. Si todo par de cadenas que conducen a estados diferentes en M , también conducen a estados diferentes en M' , entonces la cantidad de estados de M' es mayor o igual que la cantidad de estados de M :

$$\left(\forall \alpha, \beta \in \Sigma^* : \hat{\delta}(q_0, \alpha) \neq \hat{\delta}(q_0, \beta) \Rightarrow \hat{\delta}'(q'_0, \alpha) \neq \hat{\delta}'(q'_0, \beta) \right) \Rightarrow |Q| \leq |Q'|$$

Demostración. Basta con hallar una función inyectiva $f : Q \rightarrow Q'$. La existencia de dicha función implica que $|Q| \leq |Q'|$. Considerar entonces la función $g : Q \rightarrow \Sigma^*$ definida por

$$g(q) = \min \{ \alpha \in \Sigma^* : \hat{\delta}(q_0, \alpha) = q \}$$

donde se supone una relación de orden en Σ^* dada por la longitud: el orden lo define la longitud en caso de que las cadenas tengan distinta longitud, y el orden lexicográfico cuando tienen igual longitud. Así, puede definirse una función $f : Q \rightarrow Q'$ (como la de la Figura 3)

$$f(q) = \hat{\delta}'(q'_0, g(q))$$

Como para cualquier par de estados distintos $p, q \in Q$ vale que

$$\hat{\delta}(q_0, g(p)) \neq \hat{\delta}(q_0, g(q))$$

, entonces también vale que

$$\hat{\delta}'(q'_0, g(p)) \neq \hat{\delta}'(q'_0, g(q))$$

Pero esto, por definición de f , es lo mismo que

$$f(p) \neq f(q)$$

Por lo tanto, f es inyectiva, con lo cual se concluye que $|Q| \leq |Q'|$. \square

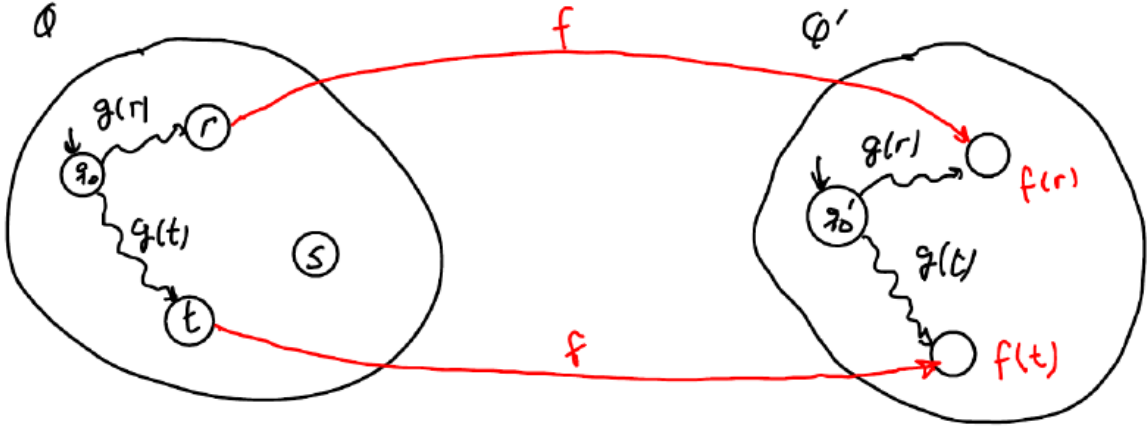


Figura 3: Función $f : Q \rightarrow Q'$ del Lema 2.1

Lema 2.2. Sea $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ un AFD, y sea $M_R = \langle Q_R, \Sigma, \delta_R, q_0, F \rangle$ el AFD reducido (mediante el algoritmo de minimización) correspondiente a M . Entonces, cualquier AFD $M' = \langle Q', \Sigma, \delta', q'_0, F \rangle$ que reconozca el mismo lenguaje que M_R , no tendrá menos estados que M_R :

$$\forall M' : \mathcal{L}(M') = \mathcal{L}(M_R) \Rightarrow |Q'| \geq |Q_R|$$

Demostración. Procediendo por el absurdo: suponer que existe un AFD $M' = \langle Q', \Sigma, \delta', q'_0, F \rangle$ tal que $\mathcal{L}(M') = \mathcal{L}(M_R)$ pero $|Q'| < |Q_R|$. Luego, por el Lema 2.1 existen dos cadenas $\alpha, \beta \in \Sigma^*$ tales que:

$$(\widehat{\delta}_R(q_0, \alpha) \neq \widehat{\delta}_R(q_0, \beta)) \wedge (\hat{\delta}'(q'_0, \alpha) = \hat{\delta}'(q'_0, \beta))$$

Luego, como $\widehat{\delta}_R(q_0, \alpha)$ y $\widehat{\delta}_R(q_0, \beta)$ son distinguibles (porque pertenecen a los estados de M_R , el AFD construido mediante el algoritmo de minimización para M), entonces, por definición de distinguibilidad:

$$\exists \gamma \in \Sigma^* \mid \widehat{\delta}_R(q_0, \alpha\gamma) \in F \wedge \widehat{\delta}_R(q_0, \beta\gamma) \notin F \quad \text{o viceversa.}$$

Por lo tanto, se deduce que:

$$\alpha\gamma \in \mathcal{L}(M_R) \Leftrightarrow \beta\gamma \notin \mathcal{L}(M_R) \quad (6)$$

Por otro lado, como $\hat{\delta}'(q'_0, \alpha) = \hat{\delta}'(q'_0, \beta)$, es claro por la definición de las funciones de transición en AFDs, que $\hat{\delta}'(q'_0, \alpha\gamma)$ y $\hat{\delta}'(q'_0, \beta\gamma)$ son ambos finales o ninguno lo es, es decir:

$$\hat{\delta}'(q'_0, \alpha\gamma) \in F \Leftrightarrow \hat{\delta}'(q'_0, \beta\gamma) \in F$$

Por lo tanto, por definición de cadena aceptada por un AFD, se deduce que:

$$\alpha\gamma \in \mathcal{L}(M') \Leftrightarrow \beta\gamma \notin \mathcal{L}(M') \quad (7)$$

Ahora viendo (6) y (7), se concluye que:

$$\mathcal{L}(M_R) \neq \mathcal{L}(M')$$

, lo cual contradice la hipótesis inicial, que era $\mathcal{L}(M_R) = \mathcal{L}(M')$. Esto es absurdo, y proviene de haber supuesto que existía un AFD M' tal que $|Q'| < |Q_R|$. Luego, dicho AFD no existe. \square

Con estos resultados, se demuestra que el algoritmo de minimización estudiado efectivamente obtiene un AFD equivalente con la mínima cantidad de estados posible, manteniendo el lenguaje reconocido.

2.5. Equivalencia entre Autómatas Finitos y Gramáticas Regulares

Se verán dos teoremas, que probarán que los autómatas finitos (AF) reconocen los mismos lenguajes que generan las gramáticas regulares (GR).

Teorema 2.7. *Dada una gramática regular (GR) $G = \langle V_N, V_T, P, S \rangle$, existe un autómata finito no determinístico (AFND) $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ tal que $\mathcal{L}(G) = \mathcal{L}(M)$.*

Demostración. Se define el AFND M de la siguiente manera:

- $Q = V_N \cup \{q_f\}$, llamando q_A al estado correspondiente al símbolo no terminal A
- $\Sigma = V_T$
- $q_0 = q_S$ (el estado de M correspondiente al símbolo distinguido S de G)
- $q_B \in \delta(q_A, a) \Leftrightarrow (A \rightarrow a B) \in P$
- $q_f \in \delta(q_A, a) \Leftrightarrow (A \rightarrow a) \in P$
- $q_A \in F \Leftrightarrow (A \rightarrow \lambda) \in P$
- $q_f \in F$

Como paso intermedio, se probará que:

$$(A \xRightarrow{*} x B) \Leftrightarrow (q_B \in \delta(q_A, x))$$

Para ello, se hará inducción en el largo de la cadena x :

- Caso base: $x = \lambda$. Hay que probar que $(A \xRightarrow{*} \lambda B) \Leftrightarrow (q_B \in \delta(q_A, \lambda))$, es decir:

$$(A \xRightarrow{*} A) \Leftrightarrow (q_A \in \delta(q_A, \lambda))$$

, lo cual es trivialmente cierto.

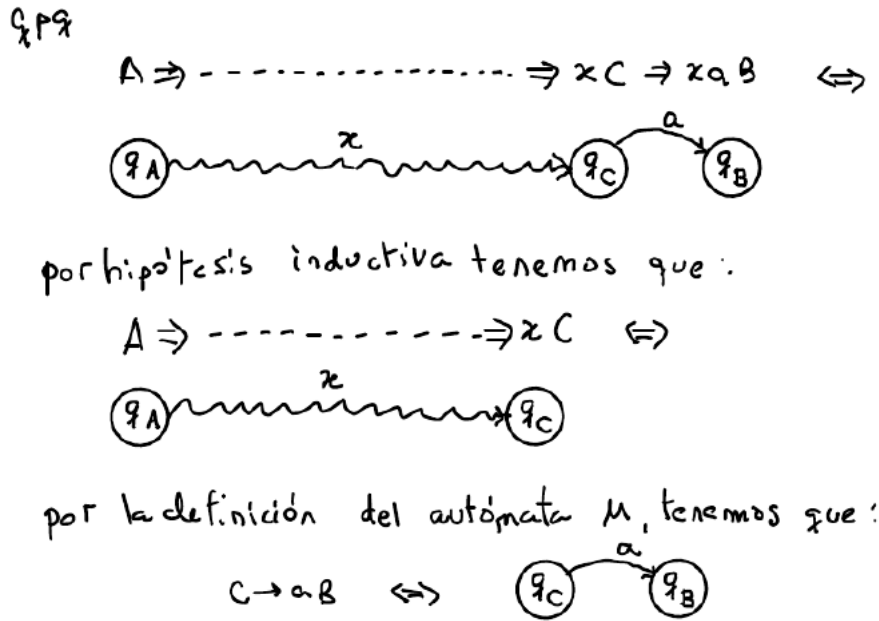
- Paso inductivo: suponiendo que vale el resultado para x de largo $|x|$, hay que probarlo para xa , de largo $|x| + 1$:

$$\begin{aligned}
 A \xRightarrow{*} x a B &\Leftrightarrow \exists C \in V_N \mid A \xRightarrow{*} x C \wedge C \rightarrow a B, \text{ por definición de derivación} \\
 &\Leftrightarrow \exists q_C \in Q \mid q_C \in \delta(q_A, x) \wedge q_B \in \delta(q_C, a), \text{ por h.i., y por definición de } M \\
 &\Leftrightarrow q_B \in \delta(\delta(q_A, x), a), \text{ por definición de } \delta \\
 &\Leftrightarrow q_B \in \delta(q_A, xa), \text{ por definición de } \delta \text{ extendida}
 \end{aligned}$$

Con eso queda probado el resultado intermedio: para toda cadena x , vale que

$$(A \xRightarrow{*} x B) \Leftrightarrow (q_B \in \delta(q_A, x))$$

En la siguiente imagen se puede ver de forma gráfica el razonamiento seguido:



Utilizando lo anterior, ver que: para una cadena w y un símbolo a

$$\begin{aligned}
 wa \in \mathcal{L}(G) &\Leftrightarrow S \xRightarrow{*} wa, \text{ por definición de lenguaje generado por } G \\
 &\Leftrightarrow (\exists A \in V_N \mid S \xRightarrow{*} wA \wedge A \rightarrow a \in P) \vee (\exists B \in V_N \mid S \xRightarrow{*} waB \wedge B \rightarrow \lambda \in P), \\
 &\quad \text{porque esas son las dos únicas maneras de obtener una forma sentencial formada} \\
 &\quad \text{sólo por símbolos terminales} \\
 &\Leftrightarrow (\exists q_A \in Q \mid q_A \in \delta(q_S, w) \wedge q_f \in \delta(q_A, a)) \vee (\exists q_B \in Q \mid q_B \in \delta(q_S, wa) \wedge q_B \in F), \\
 &\quad \text{por el resultado intermedio, y por definición del AFND } M \\
 &\Leftrightarrow q_f \in \delta(q_S, wa) \vee (\exists q_B \in Q \mid q_B \in \delta(q_S, wa) \wedge q_B \in F), \\
 &\quad \text{por definición de } \delta \text{ extendida} \\
 &\Leftrightarrow wa \in \mathcal{L}(M), \text{ por definición de cadena aceptada por un AFND}
 \end{aligned}$$

Por último, falta considerar el caso de la cadena nula λ (ya que lo visto recién era para cadenas de largo mayor o igual a 1):

$$\begin{aligned}
 \lambda \in \mathcal{L}(G) &\Leftrightarrow S \xRightarrow{*} \lambda, \text{ por definición de lenguaje generado por } G \\
 &\Leftrightarrow S \rightarrow \lambda \in P \\
 &\Leftrightarrow q_S \in F, \text{ por definición del AFND } M \\
 &\Leftrightarrow \lambda \in \mathcal{L}(M), \text{ por definición de cadena aceptada por un AFND}
 \end{aligned}$$

Así, queda probado que para toda cadena x , vale:

$$x \in \mathcal{L}(G) \Leftrightarrow x \in \mathcal{L}(M)$$

□

Observación. A partir del Teorema 2.7, y usando los Teoremas 2.1 y 2.2, se tiene que:

$$\text{GR} \Rightarrow \text{AF}$$

siendo AF todos los tipos de autómatas finitos estudiados (AFD, AFND, AFND- λ).

En el siguiente teorema, se verá la recíproca de la última implicación.

Teorema 2.8. *Dado un autómata finito determinístico (AFD) $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, existe una gramática regular (GR) $G = \langle V_N, V_T, P, S \rangle$ tal que $\mathcal{L}(M) = \mathcal{L}(G)$.*

Demostración. Se define la gramática G de la siguiente manera:

- $V_N = Q$, llamando A_p al símbolo no terminal correspondiente al estado $p \in Q$
- $V_T = \Sigma$
- $S = A_{q_0}$
- $A_p \rightarrow a A_q \in P \Leftrightarrow \delta(p, a) = q$
- $A_p \rightarrow a \in P \Leftrightarrow \delta(p, a) = q \in F$
- $S \rightarrow \lambda \in P \Leftrightarrow q_0 \in F$

Como paso intermedio, se probará que:

$$(\delta(p, w) = q) \Leftrightarrow (A_p \xRightarrow{*} w A_q)$$

Para ello, se hará inducción en el largo de la cadena w :

- Caso base: $w = \lambda$. Hay que probar que $\delta(p, \lambda) = q \Leftrightarrow (A_p \xRightarrow{*} \lambda A_q)$, es decir:

$$\delta(p, \lambda) = p \Leftrightarrow (A_p \xRightarrow{*} A_p)$$

, lo cual es trivialmente cierto.

- Paso inductivo: suponiendo que vale el resultado para x de largo $|x|$, hay que probarlo para $w = xa$, de largo $|x| + 1$:

$$\delta(p, xa) = q \Leftrightarrow \exists r \in Q \mid \delta(p, x) = r \wedge \delta(r, a) = q, \text{ por definición de } \delta \text{ extendida}$$

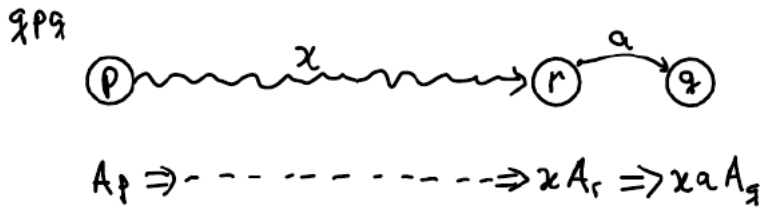
$$\Leftrightarrow \exists A_r \in V_N \mid A_p \xRightarrow{*} x A_r \wedge A_r \rightarrow a A_q \in P, \text{ por h.i., y por definición de } G$$

$$\Leftrightarrow A_p \xRightarrow{*} x a A_q, \text{ por definición de derivación}$$

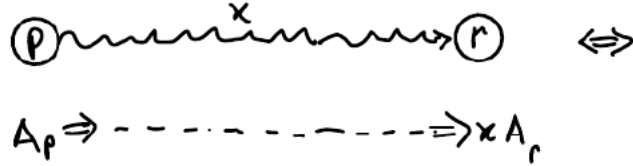
Con eso queda probado el resultado intermedio: para toda cadena w , vale que

$$\delta(p, w) = q \Leftrightarrow (A_p \xRightarrow{*} w A_q)$$

En la siguiente imagen se puede ver de forma gráfica el razonamiento seguido:



por hipótesis inductiva, tenemos que:



por la definición de la gramática G , tenemos que:



Utilizando lo anterior, ver que: para una cadena w y un símbolo a

$$\begin{aligned}
 wa \in \mathcal{L}(M) &\Leftrightarrow \delta(q_0, wa) \in F, \text{ por definición de cadena aceptada por un AFD} \\
 &\Leftrightarrow \exists p \in Q \mid \delta(q_0, w) = p \wedge \delta(p, a) \in F, \text{ por definición de } \delta \text{ extendida} \\
 &\Leftrightarrow \exists A_p \mid A_{q_0} \xRightarrow{*} w A_p \wedge A_p \rightarrow a \in P, \text{ por el resultado intermedio, y por definición de } G \\
 &\Leftrightarrow A_{q_0} \xRightarrow{*} w a, \text{ por definición de derivación} \\
 &\Leftrightarrow wa \in \mathcal{L}(G), \text{ por definición de cadena generada por una gramática}
 \end{aligned}$$

Por último, falta considerar el caso de la cadena nula λ (ya que lo visto recién era para cadenas de largo mayor o igual a 1):

$$\begin{aligned}
 \lambda \in \mathcal{L}(M) &\Leftrightarrow q_0 \in F, \text{ por definición de cadena aceptada por un AFD} \\
 &\Leftrightarrow S \rightarrow \lambda \in P, \text{ por definición de } G \\
 &\Leftrightarrow S \xRightarrow{*} \lambda, \text{ por definición de derivación} \\
 &\Leftrightarrow \lambda \in \mathcal{L}(G), \text{ por definición cadena generada por una gramática}
 \end{aligned}$$

Así, queda probado que para toda cadena w , vale:

$$w \in \mathcal{L}(M) \Leftrightarrow w \in \mathcal{L}(G)$$

□

2.6. Fuentes

- Julio Jacobo. Teoría de Lenguajes, Clase Teórica 2. Primer cuatrimestre, 2022.
- Julio Jacobo. Teoría de Lenguajes, Clase Teórica 3. Primer cuatrimestre, 2022.
- Julio Jacobo. Teoría de Lenguajes, Clase Teórica 4. Primer cuatrimestre, 2022.
- Ariel Arbiser. Teoría de Lenguajes, Clase Práctica 2. Primer cuatrimestre, 2022.
- Elisa Orduna. Teoría de Lenguajes, Clase Práctica 3. Primer cuatrimestre, 2022.

3. Expresiones Regulares

Definición 3.1. Las **expresiones regulares** (ER) son cadenas de caracteres que denotan ciertos lenguajes. Se definen de la siguiente manera:

- \emptyset es una expresión regular que denota el lenguaje vacío \emptyset .
- λ es una expresión regular que denota el lenguaje $\{\lambda\}$.
- Para cada $a \in \Sigma$, a es una expresión regular que denota el lenguaje $\{a\}$.
- Si r y s denotan los lenguajes R y S respectivamente, entonces:
 - $r|s$ es una expresión regular que denota el lenguaje $R \cup S$ (unión).
 - rs es una expresión regular que denota el lenguaje RS (concatenación).
 - r^* es una expresión regular que denota el lenguaje R^* (clausura de Kleene).
 - r^+ es una expresión regular que denota el lenguaje R^+ (clausura positiva).

Notación: si r es una expresión regular que denota el lenguaje R , se nota $\mathcal{L}(r) = R$.

3.1. Ecuaciones de Expresiones Regulares

Estas son ecuaciones donde la incógnita es una expresión regular.

Ejemplo 3.1. $r = 10r + 1$

Se puede ver que $r = (10)^*1$ es solución de la ecuación, ya que reemplazando por r en ella, se obtiene:

$$(10)^*1 = 10(10)^*1 + 1 = (10(10)^* + \lambda)1 = ((10)^+ + \lambda)1 = (10)^*1$$

Como regla general, para una ecuación de la forma $r = \alpha r + \beta$, se tiene que $r = \alpha^*\beta$ es solución.

3.2. Equivalencia entre Expresiones Regulares y Autómatas Finitos

Teorema 3.1. Dada una expresión regular r , existe un AFND- λ M con un solo estado final, y sin transiciones a partir del mismo, tal que $\mathcal{L}(M) = \mathcal{L}(r)$.

Demostración. Se probará por inducción en la cantidad de operadores que aparecen en la expresión regular r .

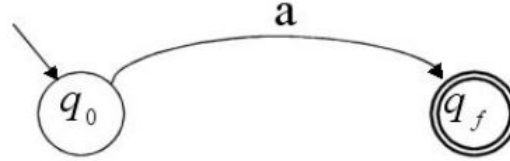
- Casos base:
 - $r = \emptyset$. El siguiente diagrama representa el autómata buscado, ya que el lenguaje definido por r es \emptyset , por lo que ninguna cadena se acepta.



- $r = \lambda$. El siguiente diagrama representa el autómata buscado, ya que el lenguaje definido por r es $\{\lambda\}$, con lo cual la única cadena que se acepta es λ .

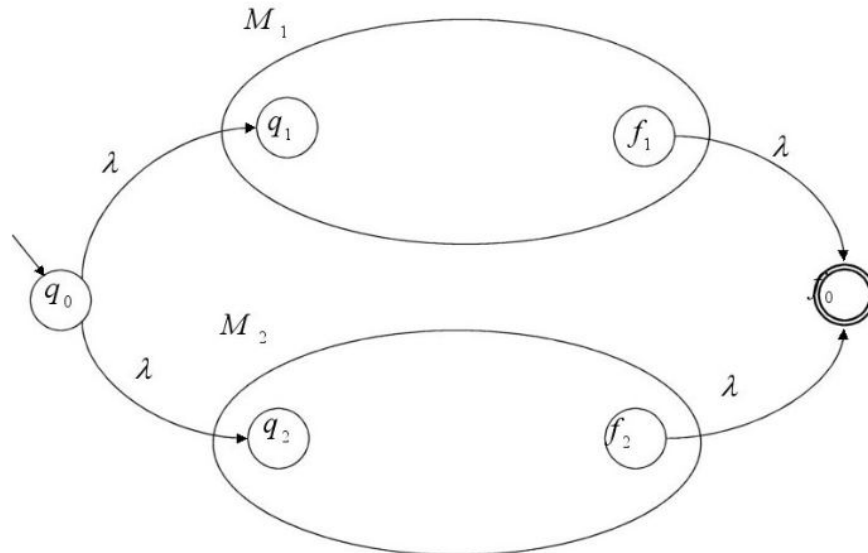


- $r = a$. El siguiente diagrama representa el autómata buscado, ya que el lenguaje definido por r es $\{a\}$, con lo cual la única cadena que se acepta es a . Entonces, desde el estado inicial q_0 , hay una transición por a al estado final q_f .



- **Paso inductivo:** suponer que vale lo que se quiere probar para las expresiones regulares r_i que están involucradas en los operadores, y se quiere probar que vale para r , que es igual a aplicar esos operadores a las r_i .
 - Caso $r = r_1|r_2$. Por hipótesis inductiva, existen los AFNDs- λ $M_1 = \langle Q_1, \Sigma_1, \delta_1, q_1, \{f_1\} \rangle$, y $M_2 = \langle Q_2, \Sigma_2, \delta_2, q_2, \{f_2\} \rangle$ tales que $\mathcal{L}(M_1) = \mathcal{L}(r_1)$ y $\mathcal{L}(M_2) = \mathcal{L}(r_2)$. Considerar el autómata $M = \langle Q_1 \cup Q_2 \cup \{q_0, f_0\}, \Sigma_1 \cup \Sigma_2, \delta, q_0, \{f_0\} \rangle$, con función de transición δ dada por:
 - $\delta(q_0, \lambda) = \{q_1, q_2\}$
 - $\delta(q, a) = \delta_1(q, a)$ para $q \in Q_1 \setminus \{f_1\}$ y $a \in \Sigma_1 \cup \{\lambda\}$
 - $\delta(q, a) = \delta_2(q, a)$ para $q \in Q_2 \setminus \{f_2\}$ y $a \in \Sigma_2 \cup \{\lambda\}$
 - $\delta(f_1, \lambda) = \{f_0\}$
 - $\delta(f_2, \lambda) = \{f_0\}$

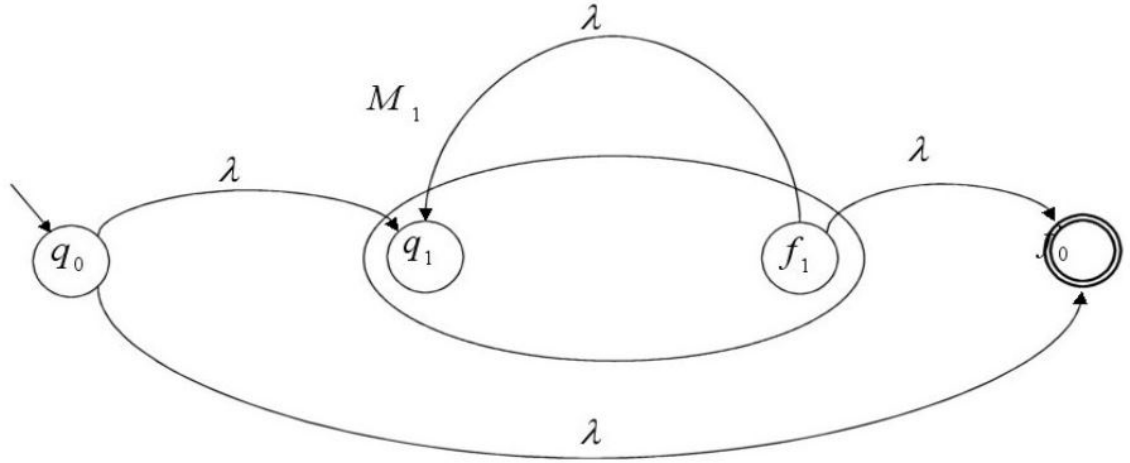
Notar que este autómata M representa la unión de los autómatas M_1 y M_2 , es decir que define el lenguaje $\mathcal{L}(M_1) \cup \mathcal{L}(M_2)$. Para eso, agrega un estado inicial q_0 , conectado mediante transiciones λ a los estados iniciales de M_1 y M_2 ; y agrega también un estado final f_0 , al que se llega desde los estados finales de M_1 y M_2 mediante transiciones λ . El siguiente diagrama representa esta situación:



- Caso $r = r_1^*$. Por hipótesis inductiva, existe un AFND- λ $M_1 = \langle Q_1, \Sigma_1, \delta_1, q_1, \{f_1\} \rangle$ tal que $\mathcal{L}(M_1) = \mathcal{L}(r_1)$. Considerar el autómata $M = \langle Q_1 \cup \{q_0, f_0\}, \Sigma_1, \delta, q_0, \{f_0\} \rangle$, con función de transición δ dada por:

- $\delta(q, a) = \delta_1(q, a)$ para $q \in Q_1 \setminus \{f_1\}$ y $a \in \Sigma_1 \cup \{\lambda\}$
- $\delta(q_0, \lambda) = \{q_1, f_0\}$
- $\delta(f_1, \lambda) = \{q_1, f_0\}$

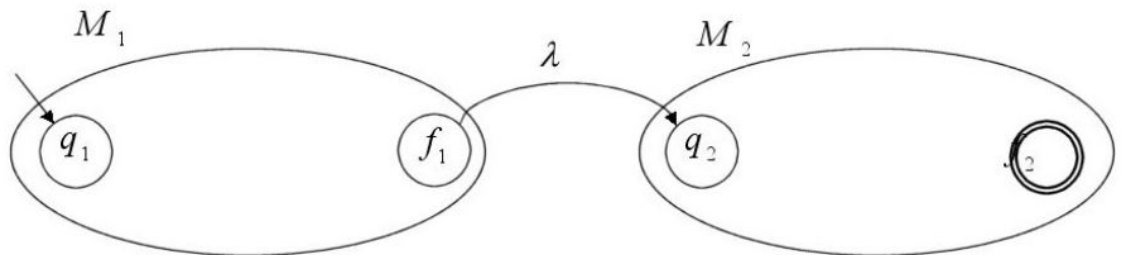
Notar que este autómata M agrega un estado inicial q_0 , desde el cual se transiciona con λ al estado inicial de M_1 . También se transiciona con λ hacia un estado final agregado f_0 , para representar que se acepta la cadena λ . Por último, desde los estados finales de M_1 hay una transición λ hacia f_0 , y además otra transición λ al estado inicial de M_1 , para representar la clausura transitiva del lenguaje. Entonces, el autómata M define el lenguaje $\mathcal{L}(M_1)^*$, y el siguiente diagrama representa su función de transición:



- Caso $r = r_1 r_2$. Por hipótesis inductiva, existen los AFNDs- λ $M_1 = \langle Q_1, \Sigma_1, \delta_1, q_1, \{f_1\} \rangle$, y $M_2 = \langle Q_2, \Sigma_2, \delta_2, q_2, \{f_2\} \rangle$ tales que $\mathcal{L}(M_1) = \mathcal{L}(r_1)$ y $\mathcal{L}(M_2) = \mathcal{L}(r_2)$. Considerar el autómata $M = \langle Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta, q_1, \{f_2\} \rangle$, con función de transición δ dada por:

- $\delta(q, a) = \delta_1(q, a)$ para $q \in Q_1 \setminus \{f_1\}$ y $a \in \Sigma_1 \cup \{\lambda\}$
- $\delta(f_1, \lambda) = \{q_2\}$
- $\delta(q, a) = \delta_2(q, a)$ para $q \in Q_2 \setminus \{f_2\}$ y $a \in \Sigma_2 \cup \{\lambda\}$

Notar que este autómata M consiste en simplemente *pegar* el final de M_1 al principio de M_2 , mediante una transición λ entre el estado final de M_1 y el estado inicial de M_2 . Es así que M reconoce la concatenación de los lenguajes reconocidos por M_1 y M_2 . Esto se puede observar en el siguiente diagrama:



Con eso quedan cubiertos todos los posibles casos para formar una expresión regular usando operadores, con lo cual se prueba el paso inductivo.

□

Se demostró en el Teorema 3.1 que para toda expresión regular, hay un autómata finito no determinístico con transiciones λ que reconoce el mismo lenguaje denotado por la expresión regular. Es decir, se tiene que:

$$ER \Rightarrow AFND-\lambda$$

Y por lo visto en los Teoremas 2.1 y 2.2, esto también significa que:

$$ER \Rightarrow AFND-\lambda \Leftrightarrow AFND \Leftrightarrow AFD$$

Para concluir la equivalencia expresiva entre expresiones regulares y autómatas finitos, falta entonces probar que para todo AF existe una ER que denota el mismo lenguaje:

Teorema 3.2. *Dado un autómata finito determinístico (AFD) $M = \langle \{q_1, \dots, q_n\}, \Sigma, \delta, q_1, F \rangle$, existe una expresión regular (ER) r tal que $\mathcal{L}(r) = \mathcal{L}(M)$.*

Demostración. Sea $R_{i,j}^k$ el conjunto de cadenas de Σ^* que llevan al autómata M desde el estado q_i al estado q_j , pasando por estados cuyo índice es a lo sumo k .

Observar que, con esta definición, $R_{i,j}^n$ es el conjunto de todas las cadenas que llevan de q_i a q_j en M , y $R_{i,j}^0$ es el conjunto de caracteres que llevan del estado q_i al estado q_j , al que se debe agregar λ en caso de que $q_i = q_j$.

Considerar la siguiente definición recursiva para $R_{i,j}^k$:

$$R_{i,j}^k = R_{i,k}^{k-1} (R_{k,k}^{k-1})^* R_{k,j}^{k-1} \cup R_{i,j}^{k-1} \text{ para } k \geq 1$$

$$R_{i,j}^0 = \begin{cases} \{a \in \Sigma : \delta(q_i, a) = q_j\} & \text{si } i \neq j \\ \{a \in \Sigma : \delta(q_i, a) = q_j\} \cup \{\lambda\} & \text{si } i = j \end{cases}$$

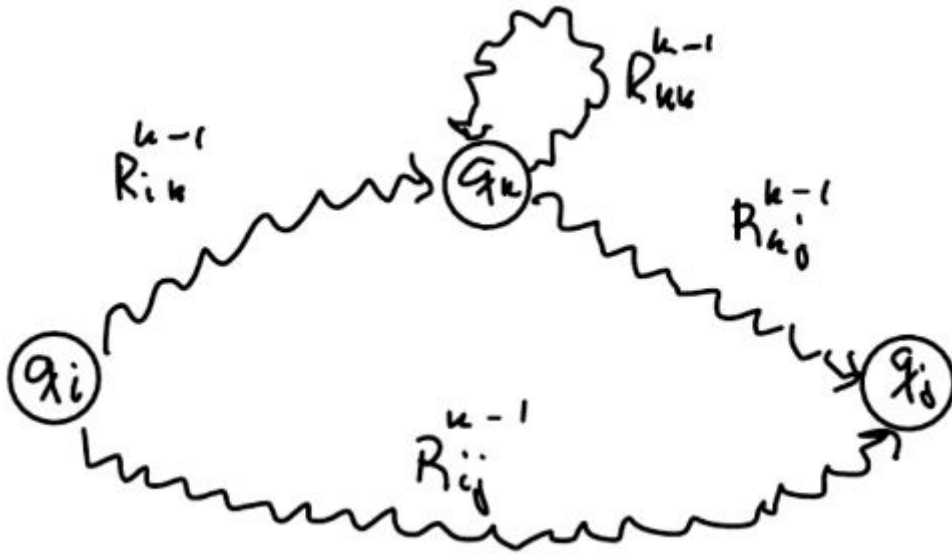
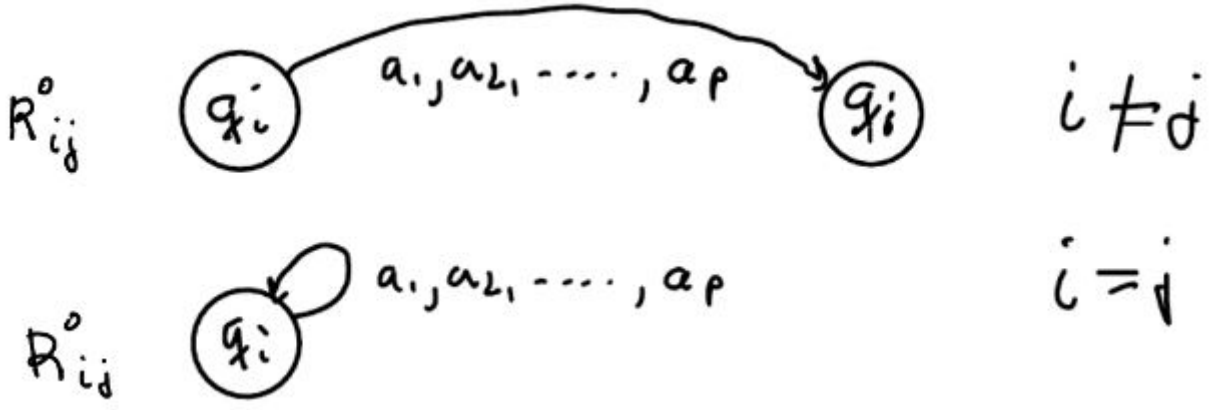


Figura 4: Caso recursivo de la definición de $R_{i,j}^k$

Como paso intermedio, se probará que hay una ER $r_{i,j}^k$ tal que $\mathcal{L}(r_{i,j}^k) = R_{i,j}^k$ para todo k , $0 \leq k \leq n$. Usando inducción en k :

Figura 5: Caso base de la definición de $R_{i,j}^k$

- Caso base: $k = 0$. Como se mencionó antes, $R_{i,j}^0$ es el conjunto de cadenas de un solo carácter, o λ . Por lo tanto, la expresión regular $r_{i,j}^0$ que lo denota será:
 - $a_1|...|a_p$, con $a_1, ..., a_p \in \Sigma$, si $\delta(q_i, a_s) = q_j$ para $s = 1, ..., p$ y $q_i \neq q_j$.
 - $a_1|...|a_p|\lambda$, con $a_1, ..., a_p \in \Sigma$, si $\delta(q_i, a_s) = q_j$ para $s = 1, ..., p$ y $q_i = q_j$.
 - \emptyset , si no existe símbolo $a_i \in \Sigma$ que conecte los estados q_i y q_j , y además $q_i \neq q_j$.
 - λ , si no existe símbolo $a_i \in \Sigma$ que conecte los estados q_i y q_j , y además $q_i = q_j$.
- Paso inductivo: por hipótesis inductiva, vale que $\mathcal{L}(r_{i,k}^{k-1}) = R_{i,k}^{k-1}$, $\mathcal{L}(r_{k,k}^{k-1}) = R_{k,k}^{k-1}$, $\mathcal{L}(r_{k,j}^{k-1}) = R_{k,j}^{k-1}$, $\mathcal{L}(r_{i,j}^{k-1}) = R_{i,j}^{k-1}$. Luego, definiendo $r_{i,j}^k$ como:

$$r_{i,j}^k = r_{i,k}^{k-1} (r_{k,k}^{k-1})^* r_{k,j}^{k-1} | r_{i,j}^{k-1}$$

, se tiene que:

$$\begin{aligned}
 \mathcal{L}(r_{i,j}^k) &= \mathcal{L}(r_{i,k}^{k-1} (r_{k,k}^{k-1})^* r_{k,j}^{k-1} | r_{i,j}^{k-1}) \\
 &= \mathcal{L}(r_{i,k}^{k-1} (r_{k,k}^{k-1})^* r_{k,j}^{k-1}) \cup \mathcal{L}(r_{i,j}^{k-1}) \\
 &= \mathcal{L}(r_{i,k}^{k-1}) \mathcal{L}((r_{k,k}^{k-1})^*) \mathcal{L}(r_{k,j}^{k-1}) \cup \mathcal{L}(r_{i,j}^{k-1}) \\
 &= \mathcal{L}(r_{i,k}^{k-1}) \mathcal{L}(r_{k,k}^{k-1})^* \mathcal{L}(r_{k,j}^{k-1}) \cup \mathcal{L}(r_{i,j}^{k-1}) \\
 &= R_{i,k}^{k-1} (R_{k,k}^{k-1})^* R_{k,j}^{k-1} \cup R_{i,j}^{k-1} \\
 &= R_{i,j}^k
 \end{aligned}$$

Con eso queda probado el resultado intermedio: $\mathcal{L}(r_{i,j}^k) = R_{i,j}^k$ para todo k , $0 \leq k \leq n$.

Por otro lado, también por lo observado al principio de la demostración:

$$\mathcal{L}(M) = \bigcup_{q_j \in F} R_{1,j}^n = R_{1,j_1}^n \cup \dots \cup R_{1,j_m}^n$$

, siendo $F = \{q_{j_1}, \dots, q_{j_m}\}$ el conjunto de estados finales del autómata M . De lo probado en la inducción, vale en particular que $\mathcal{L}(r_{1,j_i}^n) = R_{1,j_i}^n$, para $i = 1, \dots, m$.

Por lo tanto,

$$\begin{aligned}
 \mathcal{L}(M) &= \mathcal{L}(r_{1,j_1}^n) \cup \dots \cup \mathcal{L}(r_{1,j_m}^n) \\
 &= \mathcal{L}(r_{1,j_1}^n | \dots | r_{1,j_m}^n)
 \end{aligned}$$

Entonces, se concluye que el lenguaje $\mathcal{L}(M)$ es denotado por la expresión regular $r_{1,j_1}^n | \dots | r_{1,j_m}^n$. \square

Con esto, sumado a los resultados probados previamente, se tiene que:

- $\text{AFND-}\lambda \Leftrightarrow \text{AFND} \Leftrightarrow \text{AFD}$ (de los Teoremas 2.1 y 2.2).
- $\text{ER} \Rightarrow \text{AFND-}\lambda$ (del Teorema 3.1).
- $\text{AFD} \Rightarrow \text{ER}$ (del Teorema 3.2).
- $\text{GR} \Rightarrow \text{AFND}$ (del Teorema 2.7).
- $\text{AFD} \Rightarrow \text{GR}$ (del Teorema 2.8).

Con lo cual, se llegó a la equivalencia:

$$\text{ER} \Leftrightarrow \text{AFND-}\lambda \Leftrightarrow \text{AFND} \Leftrightarrow \text{AFD} \Leftrightarrow \text{GR}$$

Es decir, las expresiones regulares, los autómatas finitos y las gramáticas regulares tienen el **mismo poder expresivo**: definen los mismos lenguajes (los lenguajes regulares, o de tipo 3 en la jerarquía de Chomsky).

En la práctica, para pasar de un autómata finito a una expresión regular equivalente, primero se completaba el autómata con un *estado trampa* de ser necesario. Luego, se expresa el lenguaje mediante una ecuación, utilizando los lenguajes generados a partir de cada estado, usando el valor la función de transición para ese estado. La idea después es resolver las ecuaciones donde las incógnitas son expresiones regulares, y utilizar sustitución para despejar el valor de cada incógnita.

Para el proceso contrario (es decir, pasar de una expresión regular a un autómata finito), se utilizó el *método de las derivadas*, que consiste en observar que las ecuaciones mencionadas antes se pueden expresar en términos de las derivadas de los lenguajes generados a partir de cada estado. La idea es entonces ir derivando las expresiones regulares que definen al lenguaje generado por cada estado, para obtener una cantidad finita de expresiones regulares, que serán los estados del autómata.

3.3. Fuentes

- Julio Jacobo. Teoría de Lenguajes, Clase Teórica 3. Primer cuatrimestre, 2022.
- Ariel Arbiser. Teoría de Lenguajes, Clase Práctica 3. Primer cuatrimestre, 2022.

4. Lenguajes Regulares

4.1. Lema de pumping para Lenguajes Regulares

Idea: si las longitudes de las cadenas pertenecientes a un lenguaje L están acotadas superiormente, entonces L tiene que ser *finito*. ¿Qué condición debe cumplir el grafo de un autómata finito para que el lenguaje aceptado por éste sea *infinito*? Debe existir un camino desde el estado inicial hasta algún estado final, que pase por un ciclo (ver Figura 6). Esto es lo que captura el lema de pumping para lenguajes regulares.



Figura 6: Hay un ciclo en el camino desde el estado inicial hasta un estado final.

Lema 4.1. (Lema de pumping para lenguajes regulares) Si L es un lenguaje regular, entonces existe una longitud mínima p tal que: todas las cadenas $\alpha \in L$ con longitud mayor o igual que p pueden ser escritas de la forma $\alpha = xyz$, donde $|xy| \leq p$, $|y| \geq 1$, $\forall i \geq 0 : xy^i z \in L$. Es decir:

$$L \text{ regular} \Rightarrow \left(\exists p > 0 \mid \forall \alpha : (\alpha \in L \wedge |\alpha| \geq p) \Rightarrow \right. \\ \left. \exists x, y, z \mid (\alpha = xyz \wedge |xy| \leq p \wedge |y| \geq 1 \wedge \forall i \geq 0 : xy^i z \in L) \right)$$

Demostración. ¹ Suponer que L es un lenguaje regular. Entonces, existe un autómata finito determinístico (AFD) $A = \langle Q, \Sigma, \delta, q_0, F \rangle$ tal que $\mathcal{L}(A) = L$. Sea $n = |Q|$ la cantidad de estados de A .

Considerar una cadena $w = a_1 a_2 \dots a_m$ cualquiera, de longitud $m \geq n$. Para cada $i = 0, 1, \dots, n$, se define el estado $p_i = \hat{\delta}(q_0, a_1 a_2 \dots a_i)$, el estado en el que se encuentra A después de haber consumido los primeros i símbolos de w .

Notar que hay $n + 1$ estados p_i , pues $i = 0, 1, \dots, n$, pero hay n estados en el AFD. Por lo tanto, no es posible que todos los p_i sean distintos. Luego, existen índices i, j , con $0 \leq i < j \leq n$ tales que $p_i = p_j$.

Considerar entonces la siguiente descomposición para $w = xyz$:

- $x = a_1 a_2 \dots a_i$
- $y = a_{i+1} a_{i+2} \dots a_j$
- $z = a_{j+1} a_{j+2} \dots a_m$

En relación con los estados p_i , se puede decir que x lleva desde el estado inicial hasta p_i una vez; luego y va desde p_i hasta $p_j = p_i$ (un ciclo), y z hace el resto del camino hasta el final de w (la Figura 7 ilustra la situación). Observar que x podría ser la cadena vacía, cuando $i = 0$, y z también podría ser vacía si $j = n = m$. Sin embargo, y no puede ser vacía, ya que se tomó $i < j$.

Considerar lo que ocurre cuando el AFD A recibe la entrada xy^k , para cualquier $k \geq 0$:

¹No se dió en clase la demostración, sino sólo la idea con el diagrama del autómata. La demostración mostrada viene de: [H&U] Teorema 4.1

- si $k = 0$, entonces A va desde el estado inicial q_0 (que es igual a p_0), hasta el estado p_i al leer x . Como $p_i = p_j$, necesariamente sucede que A va desde p_i hasta el estado final que se muestra en la Figura 7 al consumir z . En consecuencia, A acepta la cadena $xy^0z = xz$.
- si $k > 0$, entonces A va desde el estado inicial $q_0 = p_0$ hasta p_i al consumir x ; luego, toma el ciclo desde p_i hasta $p_j = p_i$ una cantidad de veces igual a k , al consumir y^k . Por último, se dirige al estado final al consumir z , aceptando la cadena.

Para cualquier $k \geq 0$, se llegó a que A acepta la cadena xy^k , es decir $xy^k \in \mathcal{L}(A) = L$. \square

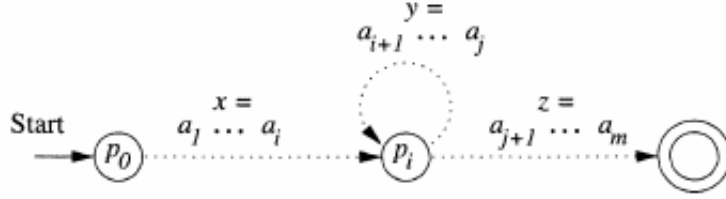


Figura 7: Recorrido por los estados p_i en la construcción del Lema 4.1. Cada cadena más larga que la cantidad de estados debe provocar que se repita un estado.

Observación. En la materia, el lema de pumping no se utiliza tal como se enuncia, sino que se utiliza el contrarrecíproco, para probar que un lenguaje dado no es regular:

Corolario. *Por el contrarrecíproco aplicado al Lema 4.1, vale que: si un lenguaje L “no cumple pumping” (no cumple el consecuente del lema de pumping), entonces L no es regular. Formalmente:*

$$\forall p > 0 : \left(\exists \alpha \left(\alpha \in L \wedge |\alpha| \geq p \wedge \right. \right. \\ \left. \left. \forall x, y, z : \left(\alpha = xyz \wedge |xy| \leq p \wedge |y| \geq 1 \wedge \exists i \geq 0 | xy^i z \notin L \right) \right) \right) \Rightarrow L \text{ no regular}$$

Conceptualmente, acerca del lema de pumping:

- Al ser L un lenguaje regular, existe un AFD con cantidad de estados mínima que reconoce L . La longitud p será la cantidad de estados de dicho autómata.
- Todas las cadenas $\alpha \in L$ cuya longitud supere la cantidad de estados del autómata ($|\alpha| > p$) hacen que en el reconocimiento de la cadena en el autómata, se pase al menos dos veces por algún estado (hay al menos un ciclo).
- Existe una descomposición $\alpha = xyz$, donde x es la parte de la cadena reconocida hasta el primer estado que se repite, y es la parte de la cadena desde dicho estado hasta la siguiente vez que se llega a él, y z es el resto de la cadena.
- Entonces, para cualquier repetición de y (recorrida del ciclo), la cadena resultante (por ejemplo, xz , xyz , $xyyz$, $xyyyz$, etc.) seguirá perteneciendo al lenguaje L . Este es el concepto de pumping o *bombeo*: siempre se puede encontrar una cadena y no vacía, y no demasiado lejana del principio de la cadena original α , tal que y se puede *bombar* (repetirla una cantidad arbitraria de veces), y que el resultado siga estando en el lenguaje.

Uso en la práctica: como se mencionó, en la materia se utiliza el contrarrecíproco del Lema de pumping para probar que un lenguaje no es regular. Los pasos que se siguen, a grandes rasgos, son:

1. El $p > 0$ viene dado, puede ser cualquiera.
2. Elegir una cadena $\alpha \in L$ tal que $|\alpha| \geq p$.
3. La descomposición $\alpha = xyz$ viene dada, puede ser cualquiera mientras cumpla que $|xy| \leq p$, e $|y| \geq 1$.
4. Elegir un $i \geq 0$ tal que $xy^iz \notin L$.
5. Concluir que L no es regular.

4.2. Unión de lenguajes regulares

La **unión** de dos lenguajes regulares resulta un lenguaje **regular**. Para ver esto, se construye el autómata finito que acepta la unión de dos lenguajes:

Dados $M_1 = \langle Q_1, \Sigma, \delta_1, q_{0_1}, F_1 \rangle$ y $M_2 = \langle Q_2, \Sigma, \delta_2, q_{0_2}, F_2 \rangle$ **autómatas finitos determinísticos** (AFD) tales que $\mathcal{L}(M_1) = L_1$ y $\mathcal{L}(M_2) = L_2$. Entonces, el autómata finito que aceptará el lenguaje $L_1 \cup L_2$ está dado por $M_\cup = \langle Q_\cup, \Sigma, \delta_\cup, q_{0_\cup}, F_\cup \rangle$, con:

- $Q_\cup = Q_1 \times Q_2$, el producto cartesiano de los conjuntos de estados de M_1 y M_2
- $\delta_\cup((q, r), a) = (\delta_1(q, a), \delta_2(r, a))$, para $q \in Q_1$ y $r \in Q_2$
- $q_{0_\cup} = (q_{0_1}, q_{0_2})$
- $F_\cup = \{(p, q) \in Q_\cup : p \in F_1 \vee q \in F_2\}$

Se puede ver que $\mathcal{L}(M_\cup) = \mathcal{L}(M_1) \cup \mathcal{L}(M_2)$, ya que:

$$\begin{aligned}
 x \in \mathcal{L}(M_\cup) &\Leftrightarrow \delta_\cup((q_{0_1}, q_{0_2}), x) \in F_\cup, \text{ por definición de cadena aceptada por un AFD} \\
 &\Leftrightarrow (\delta_1(q_{0_1}, x), \delta_2(q_{0_2}, x)) \in F_\cup, \text{ por definición de } \delta_\cup \\
 &\Leftrightarrow \delta_1(q_{0_1}, x) \in F_1 \vee \delta_2(q_{0_2}, x) \in F_2, \text{ por definición de } F_\cup \\
 &\Leftrightarrow x \in \mathcal{L}(M_1) \vee x \in \mathcal{L}(M_2), \text{ por definición de cadena aceptada por un AFD} \\
 &\Leftrightarrow x \in \mathcal{L}(M_1) \cup \mathcal{L}(M_2), \text{ por definición de unión}
 \end{aligned}$$

Esto se puede generalizar a cualquier unión finita de lenguajes regulares:

Teorema 4.1. Sean $\{L_i : i \in \mathbb{N}\}$ lenguajes regulares. Entonces:

$$\forall n \in \mathbb{N}, \bigcup_{i=1}^n L_i \text{ es regular.}$$

Demostración. Por inducción en n :

- Caso base: $n = 0$.

$$\bigcup_{i=1}^0 L_i = \emptyset, \text{ que es trivialmente regular.}$$

- Paso inductivo: suponiendo que vale para n , considerando la validez para $n + 1$:

$$\bigcup_{i=1}^{n+1} L_i \quad (8)$$

Esta expresión se puede reescribir como:

$$\bigcup_{i=1}^n L_i \cup L_{n+1}$$

El lenguaje del primer término de la unión es regular por hipótesis inductiva, y el segundo es regular por hipótesis del teorema. Luego, usando la construcción vista anteriormente para la unión de dos lenguajes regulares, se concluye que (8) es regular.

□

Observación. El resultado del teorema 4.1 no se puede generalizar a uniones infinitas. Un contraejemplo para ese hipotético resultado (que no vale) es el siguiente:

Considerar la familia de lenguajes infinita $\{L_i\}_{i=1,\infty}$. ¿Es $\bigcup_{i=1}^{\infty} L_i$ regular?

Sean $L_i = \{a^i b^i\}$. Entonces:

$$\bigcup_{i=1}^{\infty} L_i = \bigcup_{i=1}^{\infty} \{a^i b^i\} = \{a^k b^k : k \in \mathbb{N}\}$$

Y se sabe que ese lenguaje no es regular (se vio en clase).

4.3. Intersección de lenguajes regulares

La **intersección** de dos lenguajes regulares resulta un lenguaje **regular**. Para ver esto, se construye el autómata finito que acepta la intersección de dos lenguajes:

Dados $M_1 = \langle Q_1, \Sigma, \delta_1, q_{0_1}, F_1 \rangle$ y $M_2 = \langle Q_2, \Sigma, \delta_2, q_{0_2}, F_2 \rangle$ **autómatas finitos determinísticos** (AFD) tales que $\mathcal{L}(M_1) = L_1$ y $\mathcal{L}(M_2) = L_2$. Entonces, el autómata finito que aceptará el lenguaje $L_1 \cap L_2$ está dado por $M_{\cap} = \langle Q_{\cap}, \Sigma, \delta_{\cap}, q_{0_{\cap}}, F_{\cap} \rangle$, con:

- $Q_{\cap} = Q_1 \times Q_2$, el producto cartesiano de los conjuntos de estados de M_1 y M_2
- $\delta_{\cap}((q, r), a) = (\delta_1(q, a), \delta_2(r, a))$, para $q \in Q_1$ y $r \in Q_2$
- $q_{0_{\cap}} = (q_{0_1}, q_{0_2})$
- $F_{\cap} = \{(p, q) \in Q_{\cap} : p \in F_1 \wedge q \in F_2\}$

Se puede ver que $\mathcal{L}(M_{\cap}) = \mathcal{L}(M_1) \cap \mathcal{L}(M_2)$, ya que:

$$\begin{aligned} x \in \mathcal{L}(M_{\cap}) &\Leftrightarrow \delta_{\cap}((q_{0_1}, q_{0_2}), x) \in F_{\cap}, \text{ por definición de cadena aceptada por un AFD} \\ &\Leftrightarrow (\delta_1(q_{0_1}, x), \delta_2(q_{0_2}, x)) \in F_{\cap}, \text{ por definición de } \delta_{\cap} \\ &\Leftrightarrow \delta_1(q_{0_1}, x) \in F_1 \wedge \delta_2(q_{0_2}, x) \in F_2, \text{ por definición de } F_{\cap} \\ &\Leftrightarrow x \in \mathcal{L}(M_1) \wedge x \in \mathcal{L}(M_2), \text{ por definición de cadena aceptada por un AFD} \\ &\Leftrightarrow x \in \mathcal{L}(M_1) \cap \mathcal{L}(M_2), \text{ por definición de intersección} \end{aligned}$$

Esto se puede generalizar a cualquier intersección finita de lenguajes regulares:

Teorema 4.2. Sean $\{L_i : i \in \mathbb{N}\}$ lenguajes regulares. Entonces:

$$\forall n \in \mathbb{N}, \bigcap_{i=1}^n L_i \text{ es regular.}$$

Demostración. Por inducción en n :

- Caso base: $n = 0$.

$$\bigcap_{i=1}^0 L_i = \emptyset, \text{ que es trivialmente regular.}$$

- Paso inductivo: suponiendo que vale para n , considerando la validez para $n + 1$:

$$\bigcap_{i=1}^{n+1} L_i \tag{9}$$

Esta expresión se puede reescribir como:

$$\bigcap_{i=1}^n L_i \cap L_{n+1}$$

El lenguaje del primer término de la unión es regular por hipótesis inductiva, y el segundo es regular por hipótesis del teorema. Luego, usando la construcción vista anteriormente para la intersección de dos lenguajes regulares, se concluye que (9) es regular.

□

4.4. Complemento de un lenguaje regular

El conjunto de lenguajes regulares incluido en Σ^* es **cerrado respecto del complemento**:

Teorema 4.3. Sea $L = \mathcal{L}(M)$ el lenguaje regular aceptado por el autómata finito determinístico (AFD) $M = \langle Q, \Sigma, \delta, q_0, F \rangle$, siendo δ una función definida para todos los elementos del alfabeto Σ . Entonces, el autómata $M_- = \langle Q, \Sigma, \delta, q_0, Q \setminus F \rangle$ acepta el lenguaje $\Sigma^* \setminus \mathcal{L}(M)$.

Demostración. Por construcción del autómata M_- . □

Observación. Para que el autómata M_- del Teorema 4.3 esté bien definido, el autómata M debe tener todas sus transiciones definidas (si no las tenía, completarlas con *estados trampa*). La razón de esto es que si una cadena x es rechazada por M al intentar hacer una transición no definida, entonces esto mismo ocurrirá en M_- ; pero este último debería aceptar la cadena.

4.5. Lenguajes finitos

Teorema 4.4. Todo lenguaje finito es regular.

Demostración. Sean x_i las cadenas de un lenguaje L finito, con $1 \leq i \leq n$, y $n = |L|$. Se definen n lenguajes $L_i = \{x_i\}$, con lo cual L se puede reescribir de la siguiente manera:

$$L = \bigcup_{i=1}^n L_i = \bigcup_{i=1}^n \{x_i\}$$

Como cada lenguaje $\{x_i\}$ es regular, entonces, por el Teorema 4.1, L también es regular. □

4.6. Problemas decidibles acerca de lenguajes regulares

1. **Pertenencia:** *dado un lenguaje regular L sobre Σ , y $x \in \Sigma^*$, ¿pertenece x a L ?* Para responder esto, basta con realizar los siguientes pasos:

- Construir el AFD M tal que $\mathcal{L}(M) = L$.
- Si x es aceptada por M , entonces $x \in L$; si no, no.

2. **Finitud:** *dado un lenguaje regular L sobre Σ , ¿es L finito?* Un lenguaje regular L es finito si en un AFD $M = \langle Q, \Sigma, \delta, q_0, F \rangle$ que lo acepta, ningún ciclo alcanzable desde el estado inicial puede, a su vez, ser parte de un camino a algún estado final. Formalmente,

$$L \text{ finito} \Leftrightarrow \left(\forall q \in Q : \left(q_0 \xrightarrow{*} q \wedge q \xrightarrow{*} f \in F \right) \Rightarrow \left(\nexists q \xrightarrow{+} q \right) \right)$$

O equivalentemente,

$$L \text{ infinito} \Leftrightarrow \left(\exists q \in Q \mid q_0 \xrightarrow{*} q \wedge q \xrightarrow{*} f \in F \wedge q \xrightarrow{+} q \right)$$

3. **Vacuidad:** *dado un lenguaje regular L sobre Σ , ¿es L vacío?* Para responder esta pregunta:

- Construir el AFD M tal que $\mathcal{L}(M) = L$.
- Determinar el conjunto A de estados alcanzables.
- Si $F \cap A = \emptyset$, entonces L es vacío; si no, no.

4. **Equivalencia:** *dados los lenguajes regulares L_1, L_2 sobre Σ , ¿son L_1 y L_2 equivalentes?* Si el lenguaje $(L_1 \cap \overline{L_2}) \cup (\overline{L_1} \cap L_2)$ (que es regular por unión, intersección y complemento de lenguajes regulares) es vacío (lo cual se puede ver con el procedimiento descrito recién), entonces L_1 y L_2 son equivalentes; si no, no.

4.7. Fuentes

- Julio Jacobo. Teoría de Lenguajes, Clase Teórica 5. Primer cuatrimestre, 2022.
- Ariel Arbiser. Teoría de Lenguajes, Clase Práctica 6. Primer cuatrimestre, 2022.

5. Autómatas de Pila

Los autómatas de pila (AP) siguen el mismo concepto que los autómatas finitos (AF); es decir, un conjunto de estados, un alfabeto de entrada, una función de transición, estado inicial y finales, representación con un grafo, etc. Sin embargo, en los AP se agrega la posibilidad de *guardar* símbolos en una pila, y observar el tope de dicha pila (*apilar* y *desapilar*). Se verá más adelante que esto aumenta el poder expresivo en comparación a los AF; pero esto resulta relativamente intuitivo, ya que la pila añade al autómata la capacidad de *contar* la cantidad de símbolos que aparecieron en la entrada hasta cierto momento dado.

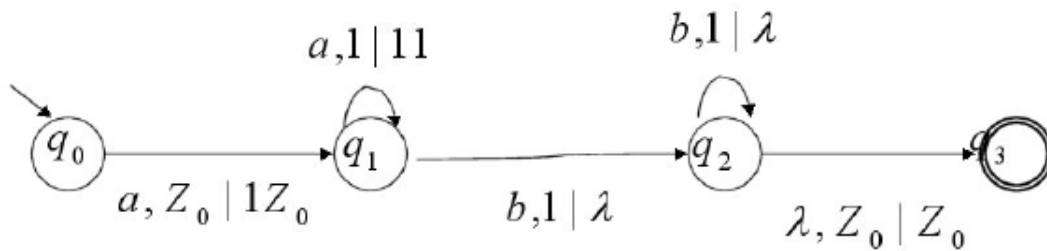
En los autómatas de pila, la **pila** es una cadena de símbolos apilados, y las transiciones entre estados dependen del símbolo que se encuentre en el tope de la pila (además del siguiente símbolo a consumir en la cadena de entrada). En cada transición, se desapila un símbolo de la pila, y se puede apilar una secuencia de símbolos.

Formalmente, la definición de los autómatas de pila no difiere demasiado de la de los autómatas finitos:

Definición 5.1. Un **autómata de pila** (AP) es una 7-upla $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$, donde:

- Q es el conjunto de estados.
- Σ es el alfabeto (es finito) de entrada.
- Γ es el alfabeto (también es finito) de pila.
- $\delta : Q \times (\Sigma \cup \{\lambda\}) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma^*)$ es la función de transición.²
- $q_0 \in Q$ es el estado inicial.
- $Z_0 \in \Gamma$ es la configuración inicial de la pila.
- $F \subseteq Q$ es el conjunto de estados finales.

Ejemplo 5.1. El siguiente grafo representa al autómata de pila que reconoce el lenguaje $L = \{a^n b^n : n \geq 1\}$



La idea es que primero se espera que venga una cadena de n apariciones del símbolo a ; por eso se apila un símbolo por cada a que se consume. Cuando llega la primera b , se transiciona a un estado distinto, en el cual se desapilará un símbolo de la pila por cada b que se consuma. Por último, se transiciona al estado final cuando no hay más cadena por consumir, y la pila está *vacía* (sólo queda Z_0 , la configuración inicial de la pila; esto es un formalismo necesario dado que no es posible realizar transiciones con la pila vacía, pues en el momento en que la pila se vacía, el autómata termina).

²Notar que devuelve conjuntos de estados, y puede tomar λ como entrada; esto es una definición similar a la del no-determinismo con transiciones λ de autómatas finitos.

Observación. Con el ejemplo anterior, ya es claro que los autómatas de pila tienen un poder expresivo distinto al de los autómatas finitos, ya que el lenguaje $L = \{a^n b^n : n \geq 1\}$ aceptado por el autómata no es regular (como se vio anteriormente).

Para abordar el tema del lenguaje aceptado por un autómata de pila, se define primero el concepto de configuración instantánea (que también se puede definir para autómatas finitos, pero no se hizo en las clases teóricas).

Definición 5.2. Una **configuración instantánea** de un autómata de pila (AP) se define como un elemento del conjunto $Q \times \Sigma^* \times \Gamma^*$, donde:

- El elemento de Q es el estado actual.
- El elemento de Σ^* es la parte de la cadena de entrada que falta procesar en el instante actual.
- El elemento de Γ^* es el contenido de la pila en el instante actual.

Observación. La configuración instantánea inicial de un AP será (q_0, α, Z_0) si la cadena de entrada es α .

Definición 5.3. Sean σ_1 y σ_2 dos configuraciones instantáneas de un autómata de pila. Las transiciones entre σ_1 y σ_2 (denotadas $\sigma_1 \vdash \sigma_2$) se definen de la siguiente manera, pudiendo ocurrir de dos maneras distintas (dependiendo de si la transición de estados es por un símbolo o por λ):

- $(q, a\alpha, t\pi) \vdash (r, \alpha, \tau\pi)$ si $(r, \tau) \in \delta(q, a, t)$
- $(q, \alpha, t\pi) \vdash (r, \alpha, \tau\pi)$ si $(r, \tau) \in \delta(q, \lambda, t)$

donde $a \in \Sigma$, $\alpha \in \Sigma^*$, $t \in \Gamma$, $\tau, \pi \in \Gamma^*$, $q, r \in Q$.

5.1. Aceptación por estado final y por pila vacía

Hay dos maneras en las que un autómata de pila puede aceptar una cadena: por estado final (como en los autómatas finitos) o por pila vacía. A continuación se definen estas dos formas.

Definición 5.4. Sea $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ un autómata de pila. Se define el **lenguaje aceptado (por estado final)** por M como:

$$\mathcal{L}(M) = \{\alpha \in \Sigma^* : (q_0, \alpha, Z_0) \vdash^* (p, \lambda, \gamma) \wedge p \in F\}$$

Definición 5.5. Sea $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ un autómata de pila. Se define el **lenguaje aceptado (por pila vacía)** por M como:

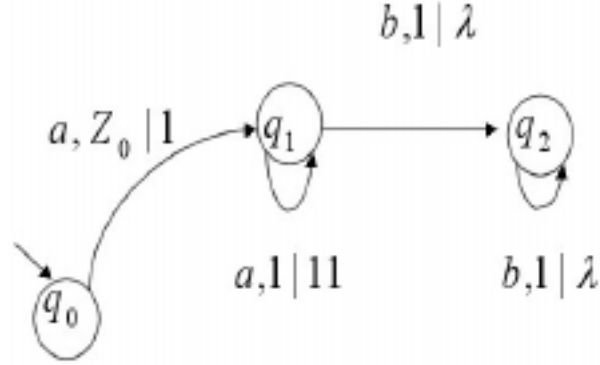
$$\mathcal{L}_\lambda(M) = \{\alpha \in \Sigma^* : (q_0, \alpha, Z_0) \vdash^* (r, \lambda, \lambda)\}$$

Es decir, una cadena es aceptada por un AP si no queda cadena por consumir y:

- se llegó a un estado final, o

- la pila está vacía.

Ejemplo 5.2. El siguiente autómata de pila acepta el mismo lenguaje que el del ejemplo 5.1, pero por pila vacía en vez de por estado final:



Se verá ahora que los lenguajes aceptados autómatas de pila por estado final son **los mismos** que los aceptados por pila vacía:

Teorema 5.1. Sea $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ un autómata de pila tal que $L = \mathcal{L}(M)$. Entonces, existe un autómata de pila M' tal que $\mathcal{L}_\lambda(M') = L$.

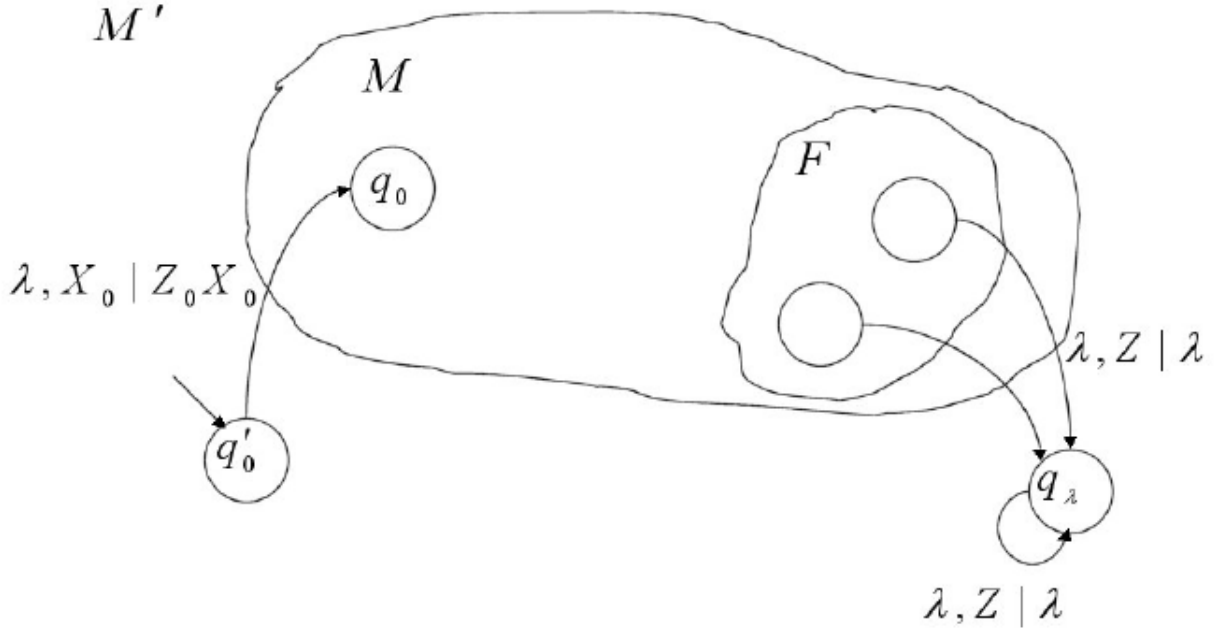


Figura 8: Grafo del autómata de pila M' construido en el Teorema 5.1

Demostración. Se define M' de la siguiente manera:

$$M' = \langle Q \cup \{q_\lambda, q'_0\}, \Sigma, \Gamma \cup \{X_0\}, \delta', q'_0, X_0, \emptyset \rangle$$

según las siguientes reglas:

1. $\delta'(q'_0, \lambda, X_0) = \{(q_0, Z_0 X_0)\}$

2. $\forall q \in Q, \forall a \in \Sigma \cup \{\lambda\} : \delta'(q, a, Z) = \{(r, \gamma) : (r, \gamma) \in \delta(q, a, Z), r \in Q \wedge \gamma \in \Gamma^*\}$
3. $\forall q \in F, \forall Z \in \Gamma \cup \{X_0\} : (q_\lambda, \lambda) \in \delta'(q, \lambda, Z)$
4. $\forall Z \in \Gamma \cup \{X_0\} : (q_\lambda, \lambda) \in \delta'(q, \lambda, Z)$

Primero, de la definición de M' se puede afirmar que:

- Por la regla 1, el autómata M' entra al estado inicial del autómata M (q_0) con Z_0X_0 en la pila, de manera que: si para una cadena cualquiera, el autómata M intenta vaciar la pila desapilando Z_0 , entonces la pila no quedaría vacía, pues estaría X_0 en el tope.
- Por la regla 2, se puede ver que M' *simula* al autómata M .
- De las reglas 3 y 4, se puede observar que si M entra en un estado final, siempre puede *elegir* entre una transición al estado q_λ y luego vaciar la pila, o bien continuar simulando M .

Ahora, hay que probar que $\mathcal{L}(M) = \mathcal{L}_\lambda(M')$. Para eso, se demostrarán las dos inclusiones por separado:

- Viendo primero que $\mathcal{L}(M) \subseteq \mathcal{L}_\lambda(M')$. Sea $x \in \mathcal{L}(M)$; entonces, por definición de lenguaje aceptado por estado final, vale que:

$$(q_0, x, Z_0) \vdash_M^* (q, \lambda, \gamma) \quad , \text{ con } q \in F.$$

Por la regla 1,

$$(q'_0, x, X_0) \vdash_{M'} (q_0, x, Z_0X_0)$$

Ahora por la regla 2,

$$(q_0, x, Z_0) \vdash_{M'} (q, \lambda, \gamma)$$

Entonces, juntando esto vale que

$$(q'_0, x, X_0) \vdash_{M'} (q_0, x, Z_0X_0) \vdash_{M'}^* (q, \lambda, \gamma X_0)$$

Y por las reglas 3 y 4, es cierto que

$$(q, \lambda, \gamma X_0) \vdash_{M'}^* (q_\lambda, \lambda, \lambda)$$

Por lo tanto,

$$(q'_0, x, X_0) \vdash_{M'}^* (q_\lambda, \lambda, \lambda)$$

, lo cual significa que si $x \in \mathcal{L}(M)$, entonces $x \in \mathcal{L}_\lambda(M')$, es decir: $\mathcal{L}(M) \subseteq \mathcal{L}_\lambda(M')$.

- Viendo ahora que $\mathcal{L}_\lambda(M') \subseteq \mathcal{L}(M)$. Sea $x \in \mathcal{L}_\lambda(M')$; entonces, existirá la secuencia de movimientos dada por

$$(q'_0, x, X_0) \vdash_{M'} (\textcolor{red}{q_0, x, Z_0X_0}) \vdash_{M'}^* (\textcolor{red}{q, \lambda, \gamma X_0}) \vdash_{M'}^* (q_\lambda, \lambda, \lambda)$$

por las reglas 1, 2, 3 y 4, respectivamente. Pero a partir de la parte en **rojo**, se puede ver que

$$(q_0, x, Z_0) \vdash_M^* (q, \lambda, \gamma)$$

Por lo tanto, si $x \in \mathcal{L}_\lambda(M')$, entonces $x \in \mathcal{L}(M)$; es decir: $\mathcal{L}_\lambda(M') \subseteq \mathcal{L}(M)$.

Con eso queda probada la igualdad $\mathcal{L}_\lambda(M') = \mathcal{L}(M)$, por lo que M y M' reconocen el mismo lenguaje (el primero por estado final, y el segundo por pila vacía). \square

Observación. En la Figura 8, se puede ver que la idea del autómata M' que se construye en el Teorema 5.1 es:

- Empezar en un estado inicial agregado, en el que se transiciona sin entrada (con λ) al estado inicial del autómata original M , apilando la configuración inicial Z_0 del mismo. Aquí se puede ver que, de cierta manera, M' *conoce la existencia* de M , pero M no conoce la de M' .
- Continuar con el comportamiento normal de M hasta llegar a sus estados finales.
- Desde los estados finales de M , salen transiciones λ a un estado q_λ *final* (no es realmente final, pero no hay transiciones a otros estados desde él).
- En el estado q_λ , se desapila símbolos hasta que la pila quede vacía, momento en el cual se acepta la cadena, ya que si se llegó hasta ese punto, es porque se alcanzaron estados finales de M sin cadena por consumir, y la pila estará vacía ahora, gracias a las transiciones *en bucle* que desapilaron todo lo que faltaba en q_λ .

Teorema 5.2. Sea $M' = \langle Q', \Sigma, \Gamma', \delta', q'_0, X_0, \emptyset \rangle$ un autómata de pila tal que $L = \mathcal{L}_\lambda(M')$. Entonces, existe un autómata de pila M tal que $\mathcal{L}(M) = L$.

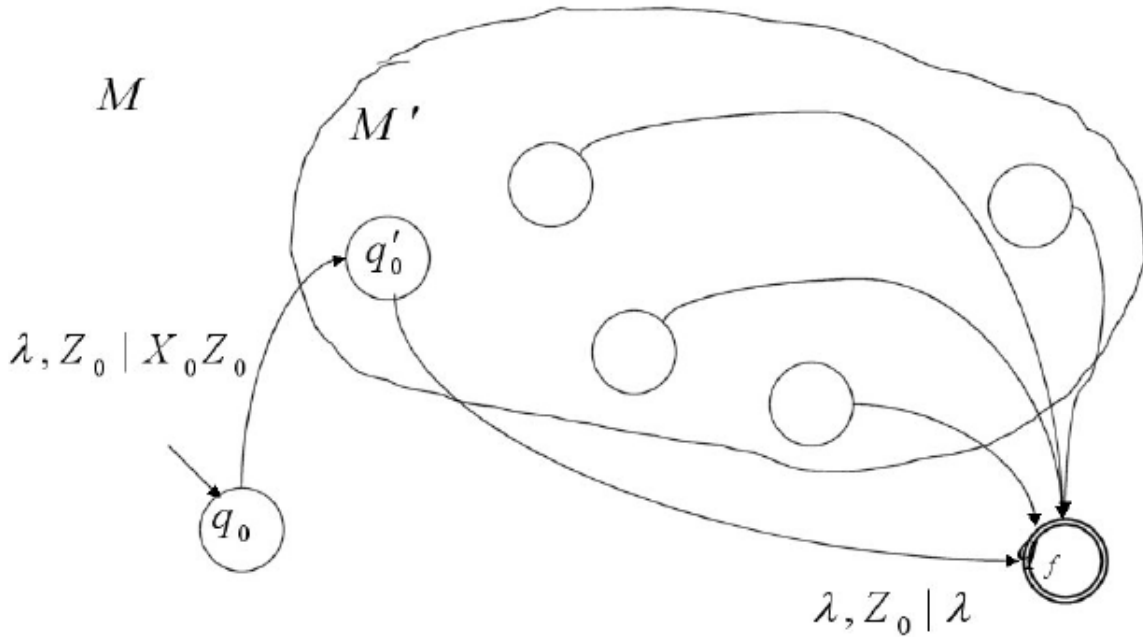


Figura 9: Grafo del autómata de pila M construido en el Teorema 5.2

Demostración. Se define M de la siguiente manera:

$$M = \langle Q \cup \{q_0, q_f\}, \Sigma, \Gamma' \cup \{Z_0\}, \delta, q_0, Z_0, \{q_f\} \rangle$$

según las siguientes reglas:

1. $\delta(q_0, \lambda, Z_0) = \{(q'_0, X_0 Z_0)\}$
2. $\forall q \in Q, \forall a \in \Sigma \cup \{\lambda\} : \delta(q, a, Z) = \delta'(q, a, Z)$
3. $\forall q \in Q : (q_f, \lambda) \in \delta(q, \lambda, Z_0)$

De la definición de M se puede afirmar que:

- Por la regla 1, el autómata M entra en M' con $X_0 Z_0$ en la pila.
- Por la regla 2, se recorre el autómata M' .
- La regla 3 permite que en cualquier momento que se vacíe la pila, pase al estado final q_f .

La demostración de que $\mathcal{L}_\lambda(M') = \mathcal{L}(M)$ es similar a la del Teorema 5.1. \square

Observación. En la Figura 9, se puede ver que la idea del autómata M que se construye en el Teorema 5.2 es:

- Empezar en un estado inicial agregado, en el que se transiciona sin entrada (con λ) al estado inicial del autómata original M' , apilando la configuración inicial X_0 del mismo.
- Continuar con el comportamiento normal de M' , hasta que se vacíe la pila, lo cual se puede detectar porque previamente a apilar X_0 , se tenía la configuración inicial Z_0 de M .
- En cuanto se vacía la pila, M puede ir al estado final q_f y aceptar la cadena, desapilando Z_0 . Es por eso que de todos los estados del autómata M' que está *incrustado* en M , hay una transición a q_f .

5.2. Relación entre Autómatas de Pila y Gramáticas Libres de Contexto

Teorema 5.3. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto (GLC) tal que $L = \mathcal{L}(G)$. Entonces, existe un autómata de pila M que acepta por pila vacía el lenguaje generado por dicha gramática, es decir: $\mathcal{L}_\lambda(M) = L = \mathcal{L}(G)$.

Demostración. ³ Se define el AP $M = \langle \{q_0\}, V_T, V_N \cup V_T, \delta, q_0, S, \emptyset \rangle$, con función de transición δ dada por:

$$\delta(q_0, a, t) = \begin{cases} \{(q_0, \alpha) : t \rightarrow \alpha \in P\} & \text{si } t \in V_N \wedge a = \lambda \\ \{(q_0, \lambda)\} & \text{si } t \in V_T \wedge a = t \neq \lambda \end{cases}$$

El autómata M acepta el lenguaje L por pila vacía porque:

- Si en el tope de la pila hay un símbolo no terminal $t \in V_N$, entonces el autómata lo reemplazará por el lado derecho α de alguna producción que tenga t en su lado izquierdo; esto lo hará de manera tal que el símbolo que esté más a la izquierda en el lado derecho de dicha producción sea el siguiente tope de la pila.
- Si en el tope de la pila hay un símbolo terminal $t \in V_T$, el autómata verificará que éste sea igual al próximo símbolo en la cadena de entrada, y lo desapilará.

³Esto no es una demostración formal completa, sino sólo la idea de la construcción. Para una demostración ver el Teorema 5.5, dado como material adicional.

□

Corolario. Por el Teorema 5.2, se deduce a partir del Teorema 5.3 que también existe un autómata de pila que acepta el lenguaje generado por la gramática libre de contexto G por estado final.

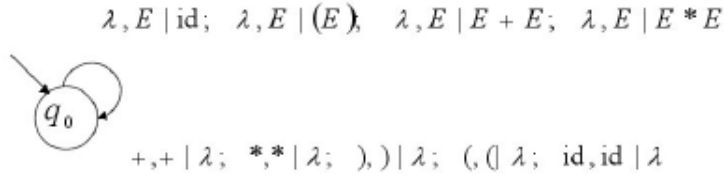
La consecuencia de esto último es que los autómatas de pila (AP) aceptan todos los lenguajes que pueden generar las gramáticas libres de contexto (GLC), sin importar si la aceptación es por pila vacía o por estado final. Es decir:

$$\text{GLC} \Rightarrow \text{AP}$$

Ejemplo 5.3. Sea $G = \langle \{E\}, \{+, *, id, (,)\}, \{E \rightarrow E + E, E \rightarrow E * E, E \rightarrow (E), E \rightarrow id\}, E \rangle$ una gramática libre de contexto que genera las expresiones aritméticas con suma y producto. El autómata de pila M que reconoce $\mathcal{L}(G)$ es:

$$M = \langle \{q_0\}, \{+, *, id, (,)\}, \{E, +, *, id, (,)\}, \delta, q_0, E, \emptyset \rangle$$

El siguiente grafo representa la función de transición de M :



Teorema 5.4. † Sea $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ un autómata de pila, y sea L el lenguaje aceptado por M por pila vacía, i.e. $L = \mathcal{L}_\lambda(M)$. Entonces L es un lenguaje libre de contexto (independiente del contexto).

Demostración. Para probar que L es libre de contexto, basta con hallar una gramática libre de contexto que lo genere. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática definida de la siguiente manera:

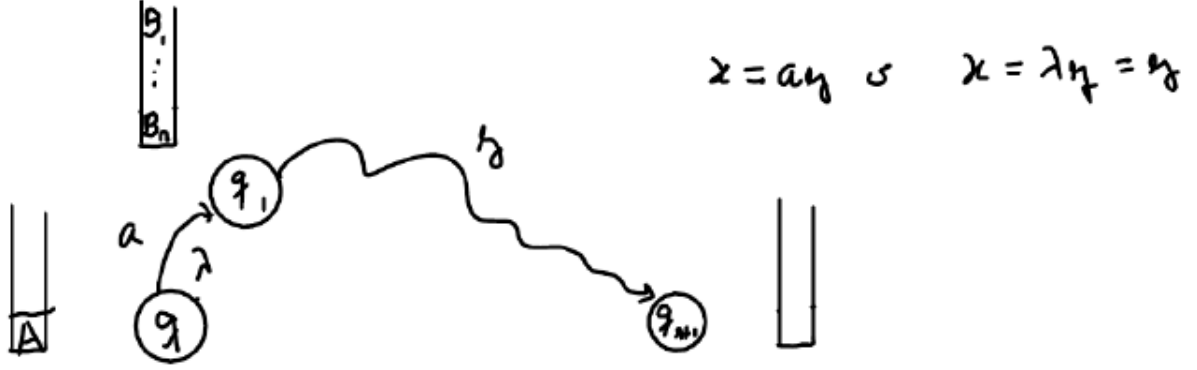
- El conjunto de símbolos no terminales es $V_N = \{[q, A, p] : q \in Q, A \in \Gamma, p \in Q \cup S\}$.
- El conjunto de símbolos terminales es $V_T = \Sigma$.
- El símbolo distinguido S es tal que $S \notin \Gamma$.
- Para cada $q, q_1, \dots, q_n \in Q$, cada $a \in \Sigma$ y cada $A, B_1, \dots, B_n \in \Gamma$, el conjunto de producciones P está dado por:

1. $S \rightarrow [q_0, Z_0, q]$ para cada $q \in Q$.
2. $[q, A, q_{n+1}] \rightarrow a[q_1, B_1, q_2] \dots [q_n, B_n, q_{n+1}] \Leftrightarrow (q_1, B_1 \dots B_n) \in \delta(q, a, A)$
3. $[q, A, q_{n+1}] \rightarrow [q_1, B_1, q_2] \dots [q_n, B_n, q_{n+1}] \Leftrightarrow (q_1, B_1 \dots B_n) \in \delta(q, \lambda, A)$
4. $[q, A, q_1] \rightarrow a \Leftrightarrow (q_1, \lambda) \in \delta(q, a, A)$
5. $[q, A, q_1] \rightarrow \lambda \Leftrightarrow (q_1, \lambda) \in \delta(q, \lambda, A)$

Al definir las producciones de esta manera, vale que:

$$[q, A, q_{n+1}] \xRightarrow{*}_G ay \Leftrightarrow (q, x, A) \vdash_M^* (q_{n+1}, \lambda, \lambda), \text{ con } x = ay$$

$$[q, A, q_{n+1}] \xRightarrow{*}_G y \Leftrightarrow (q, x, A) \vdash_M^* (q_{n+1}, \lambda, \lambda), \text{ con } x = y$$



Hay que probar que:

$$[q, A, p] \xRightarrow{*}_G x \Leftrightarrow (q, x, A) \vdash_M^* (p, \lambda, \lambda)$$



Para ello, se probará primero que para todo $i \geq 1$, vale:

$$(q, x, A) \vdash_M^i (p, \lambda, \lambda) \Rightarrow [q, A, p] \xRightarrow{*}_G x$$

Procediendo por inducción en i :

- Caso base: $i = 1$. Entonces $x = a$ o $x = \lambda$. Por lo tanto, $(q, x, A) \vdash_M^i (p, \lambda, \lambda)$ se transforma en $(q, a, A) \vdash_M^1 (p, \lambda, \lambda)$, o $(q, \lambda, A) \vdash_M^1 (p, \lambda, \lambda)$, respectivamente.

De aquí sale que $(p, \lambda) \in \delta(q, a, \lambda)$, o $(p, \lambda) \in \delta(q, \lambda, \lambda)$.

Entonces, por las reglas 4 y 5 de la definición de las producciones de la gramática, se tiene que $[q, A, p] \rightarrow a$, o bien $[q, A, p] \rightarrow \lambda$ respectivamente. Por lo tanto, vale que $(q, a, A) \vdash_M^1 (p, \lambda, \lambda) \Rightarrow [q, A, p] \xRightarrow{*}_G a$, o que $(q, \lambda, A) \vdash_M^1 (p, \lambda, \lambda) \Rightarrow [q, A, p] \xRightarrow{*}_G \lambda$, que es lo que había que probar.

- Paso recursivo: $i > 1$. Entonces, $x = ay$ con $y \in \Sigma^*$, i.e. x tiene al menos un carácter de largo. Por eso, existen $B_1, \dots, B_n \in \Gamma$ tales que $(q, ay, A) \vdash_M^i (q_1, y, B_1, \dots, B_n) \vdash_M^{i-1} (p, \lambda, \lambda)$,

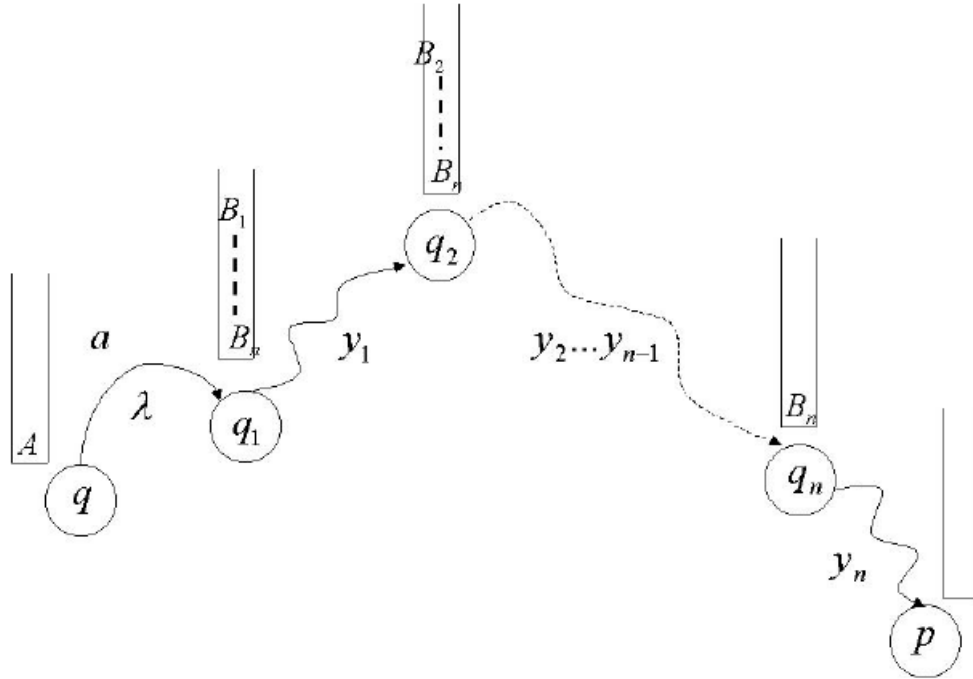
o bien $(q, y, A) \vdash_M (q_1, y, B_1, \dots, B_n) \vdash_M^{i-1} (p, \lambda, \lambda)$. Aquí B_1, \dots, B_n son símbolos de pila que reemplazan a A cuando se pasa del estado q al estado q_1 consumiendo a .

Descomponiendo la cadena y como $y = y_1 \dots y_n$, donde y_j es tal que $y_1 \dots y_j$ hace que el símbolo B_j quede en el tope de la pila por primera vez. Entonces, existen estados q_2, \dots, q_{n+1} tales que $(q_j, y_j, B_j) \vdash_M^* (q_{j+1}, \lambda, \lambda)$, para $1 \leq j \leq n$, en menos de i transiciones.

Se tiene entonces, por hipótesis inductiva, que $[q_j, B_j, q_{j+1}] \xrightarrow[G]^* y_j$ para $1 \leq j \leq n$. Con lo cual, vale que:

$$(q_1, y, B_1, \dots, B_n) \vdash_M^{i-1} (p, \lambda, \lambda) \Rightarrow [q_j, B_j, q_{j+1}] \xrightarrow[G]^* y_j$$

En la siguiente imagen se puede visualizar la situación:



Ahora, usando que:

- Por las reglas de las producciones vale que $(q, ay, A) \vdash_M (q_1, y, B_1, \dots, B_n)$, o $(q, y, A) \vdash_M (q_1, y, B_1, \dots, B_n)$.
- Se tiene entonces que $[q, A, q_{m+1}] \Rightarrow a[q_1, B_1, q_2] \dots [q_m, B_m, q_{m+1}]$ o bien $[q, A, q_{m+1}] \Rightarrow [q_1, B_1, q_2] \dots [q_m, B_m, q_{m+1}]$
- Junto con $[q_j, B_j, q_{j+1}] \xrightarrow[G]^* y_j$ para $1 \leq j \leq n$.

Se puede afirmar que $[q, A, q_{m+1}] \Rightarrow ay_1 \dots y_n = x$, o bien que $[q, A, q_{m+1}] \Rightarrow y_1 \dots y_n = x$, lo cual prueba lo que se quería.

Con eso queda probada la *ida* del resultado intermedio. Ahora, hay que demostrar la *vuelta*; es decir, que para todo $i \geq 1$:

$$[q, A, p] \xrightarrow[G]^i x \Rightarrow (q, x, A) \vdash_M^i (p, \lambda, \lambda)$$

Procediendo nuevamente por inducción en i :

- Caso base: $i = 1$. Se tiene que $[q, A, p] \xRightarrow{G}^i a$, o que $[q, A, p] \xRightarrow{G}^i \lambda$.

Esto implica que en la gramática deben existir las producciones $[q, A, p] \xRightarrow{G}^i a$, o $[q, A, p] \xRightarrow{G}^i \lambda$. Entonces, por las reglas 4 y 5 de la definición de las producciones de G , es cierto que $(p, \lambda) \in \delta(q, a, A)$ o $(p, \lambda) \in \delta(q, \lambda, A)$.

- Paso inductivo: $i > 1$. Se tiene que $[q, A, p] \xRightarrow{G} a[q_1, B_1, q_2] \dots [q_n, B_n, p] \xRightarrow{G}^{i-1} x$, o bien que $[q, A, p] \xRightarrow{G} [q_1, B_1, q_2] \dots [q_n, B_n, p] \xRightarrow{G}^{i-1} x$.

Descomponiendo a x como $x = ax_1 \dots x_n$ de manera tal que $[q_j, B_j, q_{j-1}] \xRightarrow{G}^* x_j$ con $1 \leq j \leq n$, donde cada derivación tome menos de i pasos. Por esto último, se puede aplicar la hipótesis inductiva para afirmar que $(q_j, x_j, B_j) \vdash_M^* (q_{j+1}, \lambda, \lambda)$ para $1 \leq j \leq n$.

Esto implica que $(q_j, x_j, B_j \dots B_n) \vdash_M^* (q_{j+1}, \lambda, B_{j+1} \dots B_n)$ para $1 \leq j \leq n$.

Ahora, como por lo dicho al principio del paso inductivo, $[q, A, p] \xRightarrow{G} a[q_1, B_1, q_2] \dots [q_n, B_n, p]$, o $[q, A, p] \xRightarrow{G} [q_1, B_1, q_2] \dots [q_n, B_n, p]$, entonces se tiene que $(q, a, A) \vdash_M^* (q, \lambda, B_1 \dots B_n)$, o $(q, \lambda, A) \vdash_M^* (q, \lambda, B_1 \dots B_n)$.

A partir de las siguientes cosas dichas anteriormente:

- $(q_j, x_j, B_j \dots B_n) \vdash_M^* (q_{j+1}, \lambda, B_{j+1} \dots B_n)$ para $1 \leq j \leq n$,
- $(q, a, A) \vdash_M^* (q, \lambda, B_1 \dots B_n)$, o $(q, \lambda, A) \vdash_M^* (q, \lambda, B_1 \dots B_n)$.

, y con $q_{j+1} = p$, surge que: $(q, ax_1 \dots x_n, A) \vdash_M^* (q_1, x_1 \dots x_n, B_1 \dots B_n) \vdash_M^* (p, \lambda, \lambda)$, o bien $(q, x_1 \dots x_n, A) \vdash_M^* (q_1, x_1 \dots x_n, B_1 \dots B_n) \vdash_M^* (p, \lambda, \lambda)$, que era lo que había que probar.

Así se probó también la *vuelta* del resultado intermedio. Consecuentemente, queda probado que

$$[q, A, p] \xRightarrow{G}^* x \Leftrightarrow (q, x, A) \vdash_M^* (p, \lambda, \lambda)$$

Tomando entonces $q = q_0$ y $A = Z_0$ y reemplazando en lo anterior:

$$[q_0, Z_0, p] \xRightarrow{G}^* x \Leftrightarrow (q_0, x, Z_0) \vdash_M^* (p, \lambda, \lambda)$$

Por la regla 1 de la definición de las producciones de G , existe la producción $S \rightarrow [q_0, Z_0, p]$. Con lo cual, lo anterior se transforma en:

$$S \xRightarrow{G}^* x \Leftrightarrow (q_0, x, Z_0) \vdash_M^* (p, \lambda, \lambda)$$

Y esto es lo mismo, por definición de lenguaje aceptado por un autómata de pila por pila vacía, y por definición de lenguaje generado por una gramática, que:

$$x \in \mathcal{L}(G) \Leftrightarrow x \in \mathcal{L}_\lambda(M)$$

□

Observación. El resultado probado en el Teorema 5.4 **no** es necesario para demostrar el Teorema 7.4

Ahora, en el siguiente resultado se prueba la propiedad recíproca a lo demostrado recién en el Teorema 5.4.

Teorema 5.5. † Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto, y sea L el lenguaje generado por G , i.e. $L = \mathcal{L}(G)$. Entonces, existe un autómata de pila M tal que $L = \mathcal{L}_\lambda(M)$.

Demostración. Sea $M = \langle \{q\}, V_T, V_N \cup V_T, \delta, q, S, \emptyset \rangle$, donde la función de transición δ se define como:

1. Si $A \rightarrow \alpha \in P$, entonces $(q, \alpha) \in \delta(q, \lambda, A)$.
2. $\forall a \in V_T : \delta(q, a, a) = \{(q, \lambda)\}$.

Como resultado intermedio, se probará que para algunos $m, n \geq 1$ vale que:

$$A \xRightarrow{m} w \Leftrightarrow (q, w, A) \vdash^n (q, \lambda, \lambda)$$

Viendo primero la *ida*, por inducción en m :

$$A \xRightarrow{m} w \Rightarrow (q, w, A) \vdash^n (q, \lambda, \lambda)$$

- Caso base: $m = 1$. Sea $w = a_1 \dots a_k$, con $k > 0$ (es decir, $A \Rightarrow a_1 \dots a_k$). Entonces, se tiene que:

$$(q, a_1 \dots a_k, A) \vdash (q, a_1 \dots a_k, a_1 \dots a_k) \vdash^k (q, \lambda, \lambda)$$

- Paso inductivo: $m > 1$. Suponer que vale $A \xRightarrow{m} w$. El primer paso de la derivación será de la forma $A \Rightarrow X_1 \dots X_k$, con $X_i \Rightarrow x_i$ para algún $m_i < m$, con $1 \leq i \leq k$, y donde $x_1 \dots x_k = w$.

Entonces, vale que $(q, w, A) \vdash (q, w, X_1 \dots X_k)$.

Por un lado, si $X_i \in V_N$ entonces vale, por hipótesis inductiva, que $(q, x_i, X_i) \vdash^* (q, \lambda, \lambda)$.

Por otro lado, si $X_i = x_i \in V_T$, entonces $(q, x_i, X_i) \vdash (q, \lambda, \lambda)$.

Luego, se puede afirmar que:

$$\begin{aligned} (q, w, A) &\vdash (q, x_1 \dots x_k, X_1 \dots X_k) \\ &\vdash (q, x_2 \dots x_k, X_2 \dots X_k) \\ &\dots \\ &\vdash (q, x_k, X_k) \\ &\vdash (q, \lambda, \lambda) \end{aligned}$$

Con eso queda probada la *ida* del resultado intermedio. Ahora la *vuelta*:

$$(q, w, A) \vdash^n (q, \lambda, \lambda) \Rightarrow A \xRightarrow{m} w$$

Procediendo nuevamente por inducción, esta vez en n :

- Caso base: $n = 1$. Entonces, $w = \lambda$, y $A \rightarrow \lambda \in P$, con lo cual vale el resultado.

- Paso inductivo: $n > 1$. Suponer que el resultado es cierto para todos los $n' < n$. La primera transición realizada por el autómata de pila M será de la forma $(q, w, A) \vdash (q, w, X_1 \dots X_k)$.

Además, es cierto que $(q, x_i, X_i) \vdash^{n_i} (q, \lambda, \lambda)$ para $1 \leq i \leq k$, con $w = x_1 \dots x_k$.

Entonces, como $A \rightarrow X_1 \dots X_k \in P$, y además, por hipótesis inductiva vale que si $X_i \in V_N$, entonces $X_i \xRightarrow{+} x_i$, y si $X_i \in V_T$ entonces $X_i \xRightarrow{0} x_i$. Por lo tanto, es cierto que:

$$\begin{aligned} A &\Rightarrow X_1 \dots X_k \\ &\Rightarrow x_1 X_2 \dots X_k \\ &\dots \\ &\Rightarrow x_1 \dots x_{k-1} X_k \\ &\Rightarrow x_1 \dots x_k = w \end{aligned}$$

Así queda probada también la *vuelta*, con lo cual vale el resultado intermedio:

$$A \xRightarrow{m} w \Leftrightarrow (q, w, A) \vdash^n (q, \lambda, \lambda)$$

Ahora, utilizando dicho resultado, y tomando $A = S$, se puede afirmar que:

$$S \xRightarrow{+} w \Leftrightarrow (q, w, S) \vdash^+ (q, \lambda, \lambda)$$

Por lo tanto, por la definición de lenguaje generado por una gramática, y por definición de lenguaje aceptado por un autómata de pila por pila vacía, se concluye que:

$$\mathcal{L}(G) = \mathcal{L}_\lambda(M)$$

□

Observación. La consecuencia de los Teoremas 5.4 y 5.5 es que los autómatas de pila (en realidad, los autómatas de pila no determinísticos, como se verá luego) reconocen los mismos lenguajes que los generados por las gramáticas libres de contexto. Es decir:

$$\text{AP} \Leftrightarrow \text{GLC}$$

5.3. Autómatas de Pila Determinísticos

Los autómatas de pila de los que se viene hablando son los más generales, o Autómatas de Pila No Determinísticos (APND). Pero también existen los Autómatas de Pila Determinísticos (APD); no se hace esta distinción desde el principio porque, como se verá, **los APD no son equivalentes a los APND** (estos últimos tienen mayor poder expresivo), a diferencia de lo que ocurría con los Autómatas Finitos.

Definición 5.6. Un **autómata de pila determinístico** (APD) es un autómata de pila $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$, que cumple que para todo $q \in Q, a \in \Sigma, z \in \Gamma$:

1. $|\delta(q, a, z)| \leq 1$
2. $|\delta(q, \lambda, z)| \leq 1$
3. si $|\delta(q, \lambda, z)| = 1$, entonces $|\delta(q, a, z)| = 0$

Intuitivamente, la idea de un APD es: *en cada paso, se puede tomar a lo sumo una transición.*

5.3.1. Propiedad del prefijo

Definición 5.7. Se dice que un lenguaje L posee la **propiedad del prefijo** cuando para todo par de cadenas no nulas $x, y \in L$, es cierto que $x \in L \Rightarrow xy \notin L$. En este caso también se dice que L es *libre de prefijos*.

Si un lenguaje libre de contexto *no* posee la propiedad del prefijo, entonces **todo autó-mata de pila** M que acepta L por pila vacía (es decir, $L = \mathcal{L}_\lambda(M)$) necesariamente **será no determinístico**.

Además, un autómatas de pila determinístico (APD) sí puede aceptar un lenguaje que no cumpla la propiedad del prefijo si lo hace por estados finales, con lo cual los lenguajes aceptados por APD por pila vacía no son equivalentes a los aceptados por los APD por estados finales.

Corolario. *Los autómatas de pila no determinísticos (APND) aceptan más lenguajes que los autómatas de pila determinísticos (APD), ya que los lenguajes aceptados por los APD por pila vacía son necesariamente libres de prefijos. Es decir:*

$$APD \Rightarrow APND$$

Pero:

$$APND \not\Rightarrow APD$$

Por lo tanto:

$$APD \not\Leftarrow APND$$

Eso significa que no es cierto que para cada APND existe un APD que reconoce el mismo lenguaje. Los lenguajes libres de contexto aceptados por los APD se denominan **lenguajes LR**.

5.4. Fuentes

- Julio Jacobo. Teoría de Lenguajes, Clase Teórica 5. Primer cuatrimestre, 2022.
- Julio Jacobo. Teoría de Lenguajes, Clase Teórica 6. Primer cuatrimestre, 2022.
- Verónica Becher. Teoría de Lenguajes, Clase Teórica 9. Primer cuatrimestre, 2022.
- Sabrina Silvero. Teoría de Lenguajes, Clase Práctica 7. Primer cuatrimestre, 2022.

6. Lenguajes Libres de Contexto

Definición 6.1. Una gramática $G = \langle V_N, V_T, P, S \rangle$ es **libre de contexto** (GLC) (o independiente del contexto) cuando las producciones en P son de la forma

$$A \rightarrow \alpha \quad , \text{ con } A \in V_N \text{ y } \alpha \in (V_N \cup V_T)^*$$

Si, en particular, $\alpha \in (V_N \cup V_T)^*$, es decir que no hay *reglas borradoras*, entonces se dice que la gramática G es **propia**.

Ejemplo 6.1. La gramática $G = \langle \{E\}, \{+, *, id, const\}, P, E \rangle$, con

$$P = \{ E \rightarrow E + E, \\ E \rightarrow E * E, \\ E \rightarrow id, \\ E \rightarrow const \}$$

es libre de contexto.

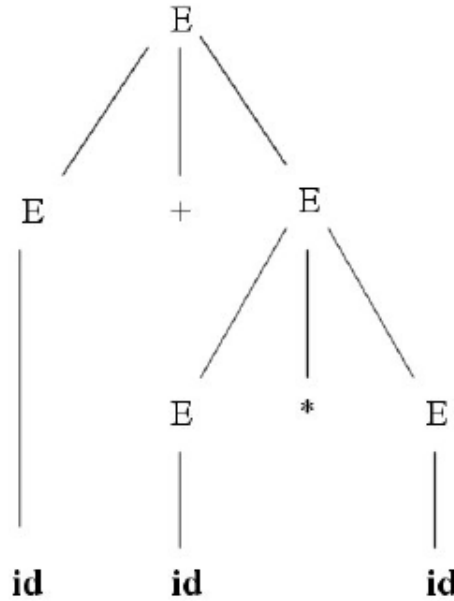


Figura 10: Posible árbol de derivación para la cadena $id + id * id$ en la gramática del ejemplo 6.1.

Definición 6.2. Una **derivación más a la izquierda** (notado \Rightarrow_L) es aquella en la que siempre se elige el símbolo no terminal que aparece más a la izquierda para reemplazar usando alguna producción.

$$\alpha_1 A \alpha_2 \Rightarrow_L \alpha_1 \alpha' \alpha_2 \text{ si } A \rightarrow \alpha' \in P, \text{ y } \alpha_1 \in V_T^*$$

Definición 6.3. Una **derivación más a la derecha** (notado \Rightarrow_R) es aquella en la que siempre se elige el símbolo no terminal que aparece más a la derecha para reemplazar usando alguna producción.

$$\alpha_1 A \alpha_2 \Rightarrow_R \alpha_1 \alpha' \alpha_2 \text{ si } A \rightarrow \alpha' \in P, \text{ y } \alpha_2 \in V_T^*$$

Relacionado con las derivaciones, existe el concepto de **análisis sintáctico** o *parsing*:

Definición 6.4. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto, y sea $\alpha \in (V_T \cup V_N)^*$. Un **parsing más a la izquierda** de α es la secuencia de producciones de G usadas en una derivación más a la izquierda de α desde S . Análogamente, un **parsing más a la derecha** de α es la secuencia de producciones de G usadas en una derivación más a la derecha de α desde S .

6.1. Árboles de derivación

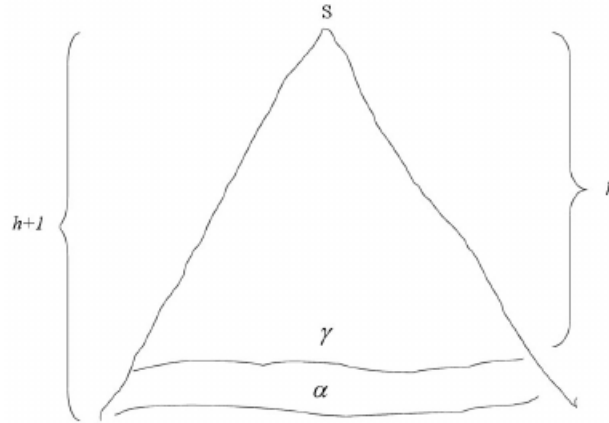
Lema 6.1. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto, con $P \neq \emptyset$, y sea $\mathcal{T}(S)$ un árbol de derivación en G para la cadena α .

Sea h la altura de $\mathcal{T}(S)$, y sea $a = \max \{ |\beta| : A \rightarrow \beta \in P \}$. Entonces:

$$|\alpha| \leq a^h$$

Demostración. Procediendo por inducción en la altura h :

- Caso base: $h = 0$. El único árbol de derivación con altura 0 posible es el que consta de sólo la raíz S . Entonces, $a^h = a^0 = 1$, y $|\alpha| = |S| = 1 \leq 1$.
- Paso inductivo: suponiendo que el resultado vale para un árbol de altura h , considerar $\mathcal{T}(S)$ con altura $h + 1$. Entonces, la longitud del camino hasta alguna de sus hojas será $h + 1$. Sea α la base de $\mathcal{T}(S)$, y sea γ la base de altura h , como muestra la siguiente figura:



Entonces, $|\alpha| \leq a|\gamma|$. Por hipótesis inductiva, se tiene que $|\gamma| \leq a^h$, con lo cual $|\alpha| \leq a|\gamma| \leq aa^h = a^{h+1}$.

□

Observación. Notar que la base de un árbol de derivación es la cadena derivada.

6.2. Ambigüedad

Definición 6.5. Una gramática libre de contexto G es **ambigua** cuando existe una cadena en el lenguaje generado por G que posee más de un árbol de derivación.

Equivalentemente, G es ambigua si tiene más de una derivación *más a la izquierda*, o más de una derivación *más a la derecha*.

Recíprocamente, una gramática libre de contexto G es *no ambigua* cuando toda cadena perteneciente al lenguaje generado por G tiene una única derivación más a la izquierda (o más a la derecha).

Observación. Acerca de la ambigüedad:

- El problema de decidir si una gramática libre de contexto es ambigua **no es decidible** (no existe un algoritmo capaz de analizar la gramática, y contestar positiva o negativamente).
- Cualquier lenguaje no vacío admite una gramática ambigua que lo genere: basta con tomar una gramática no ambigua, e introducir una regla duplicada.

La ambigüedad es una propiedad que no suele ser deseable si el objetivo es reconocer cadenas pertenecientes a una gramática de manera automatizada. Por eso, se dedica parte de la materia a estudiar formas de eliminar ambigüedad en gramáticas libres de contexto (modificar gramáticas para eliminar estas ambigüedades). Sin embargo, existen algunos lenguajes para los cuales no es posible dar con una gramática no ambigua: los lenguajes intrínsecamente ambiguos.

Observación. Acerca de la ambigüedad:

- El problema de decidir si una gramática libre de contexto es ambigua **no es decidible** (no existe un algoritmo capaz de analizar la gramática, y contestar positiva o negativamente).
- Cualquier lenguaje no vacío admite una gramática ambigua que lo genere: basta con tomar una gramática no ambigua, e introducir una regla duplicada.

Definición 6.6. Un lenguaje libre de contexto L es **intrínsecamente ambiguo** cuando toda gramática libre de contexto G tal que $L = \mathcal{L}(G)$ es ambigua.

Ejemplo 6.2. El siguiente lenguaje es libre de contexto, por ser unión de lenguajes libres de contexto, y es intrínsecamente ambiguo:

$$\{a^n b^m c^m d^n \mid n, m > 0\} \cup \{a^n b^n c^m d^m \mid n, m > 0\}$$

Para tratar con una gramática ambigua, se puede:

- Intentar cambiar la gramática para que deje de ser ambigua (ver 6.3 para más detalles).
- Descartar árboles de derivación, dando reglas de precedencia.

6.3. Escritura y Reducción de Gramáticas Libres de Contexto

Recordar la definición de forma sentencial:

Definición 6.7. Dada una gramática $G = \langle V_N, V_T, P, S \rangle$, una cadena $\alpha \in (V_T \cup V_N)^*$ se denomina **forma sentencial** cuando $S \xRightarrow{*} \alpha$.

6.3.1. Símbolos útiles

Definición 6.8. Dada una gramática $G = \langle V_N, V_T, P, S \rangle$, un símbolo no terminal $A \in V_N$ se dice **alcanzable** cuando existe una forma sentencial que lo contiene. Es decir, cuando $S \xRightarrow{*} \alpha A \beta$, con $\alpha, \beta \in (V_T \cup V_N)^*$. Se dice que A es *inalcanzable* si no es alcanzable.

Definición 6.9. Dada una gramática $G = \langle V_N, V_T, P, S \rangle$, un símbolo no terminal $A \in V_N$ se dice **activo** cuando A deriva en cero o más pasos en una cadena de símbolos terminales (o λ); es decir, cuando $\exists x \in V_T^* \mid A \xRightarrow{*} x$. Se dice que A es *inactivo* si no es activo.

Definición 6.10. Dada una gramática $G = \langle V_N, V_T, P, S \rangle$, un símbolo no terminal $A \in V_N$ se dice **útil** cuando es parte de una forma sentencial que genera una cadena de símbolos terminales (i.e. cuando A es alcanzable y activo). Es decir, cuando $S \xRightarrow{*} \alpha A \beta \xRightarrow{*} w$, con $w \in V_T^*$, $\alpha, \beta \in (V_T \cup V_N)^*$. Se dice que A es *inútil* si no es útil.

Existen algoritmos para eliminar los símbolos inútiles, que se basan en calcular los conjuntos de símbolos activos y símbolos alcanzables, para luego eliminar las producciones que incluyen a los símbolos que no están contenidos en dichos conjuntos. Los algoritmos estudiados terminan cuando, en una iteración, el conjunto obtenido es el mismo que en el paso anterior.

Algoritmo para encontrar símbolos activos

```

1:  $N \leftarrow \emptyset$ 
2: repeat
3:    $N \leftarrow N \cup \{A \mid A \rightarrow \alpha \in P \wedge \alpha \in (N \cup V_T)^*\}$ 
4: until  $N$  no cambió

```

Algoritmo para encontrar símbolos alcanzables

```

1:  $N \leftarrow \{S\}$ 
2: repeat
3:    $N \leftarrow N \cup \{X \mid A \rightarrow \alpha X \beta \in P \wedge A \in N\}$ 
4: until  $N$  no cambió

```

Entonces, para eliminar los símbolos inútiles: primero eliminar símbolos inactivos, y luego eliminar símbolos inalcanzables. Se debe hacer en ese orden porque puede ocurrir que un símbolo fuera alcanzable a través de uno inactivo. En cambio, si un símbolo utiliza otros inalcanzables para *ser activo*, entonces él mismo era inalcanzable.

Definición 6.11. Una gramática $G = \langle V_N, V_T, P, S \rangle$ se dice **reducida** si todo símbolo no terminal $A \in V_N$ es útil (alcanzable y activo).

6.3.2. Símbolos anulables

Definición 6.12. Dada una gramática $G = \langle V_N, V_T, P, S \rangle$, un símbolo no terminal $A \in V_N$ se dice **anulable** cuando genera λ , es decir cuando $A \xRightarrow{*} \lambda$.

Algoritmo para encontrar símbolos anulables

```

1:  $N \leftarrow \{A \mid A \rightarrow \lambda \in P\}$ 
2: repeat
3:    $N \leftarrow N \cup \{A \mid A \rightarrow \alpha \in P \wedge \alpha \in N^*\}$ 
4: until N no cambió

```

Definición 6.13. Una gramática $G = \langle V_N, V_T, P, S \rangle$ se dice **propia** si no existen reglas borradoras en P , es decir: $\forall A \rightarrow \alpha \in P : \alpha \neq \lambda$, con $A \in V_N$ y $\alpha \in (V_N \cup V_T)^*$.

Proposición 6.1. Sea G una gramática libre de contexto. Existe una **gramática propia** (sin reglas borradoras) G' que genera el mismo lenguaje que G , sin la cadena nula; es decir: $\mathcal{L}(G) = \mathcal{L}(G')$ o $\mathcal{L}(G) = \mathcal{L}(G') \cup \{\lambda\}$.

Calcular los símbolos anulables sirve para poder eliminar las **reglas borradoras** (también llamadas producciones λ). Para eso, se debe:

1. Tomar cada producción de la forma $B \rightarrow \alpha_1 A_1 \alpha_2 A_2 \alpha_3 \dots \alpha_k A_k \alpha_{k+1}$, con $k \geq 1$ y A_1, A_2, \dots, A_k símbolos anulables.
2. Reemplazar esas producciones por reglas de la forma $B \rightarrow \alpha_1 \beta_1 \alpha_2 \beta_2 \alpha_3 \dots \alpha_k \beta_k \alpha_{k+1}$ tales que $\beta_j = A_j \vee \beta_j = \lambda \forall 1 \leq j \leq k$.
3. Eliminar cada regla de la forma $A_j \rightarrow \lambda$, salvo por $S \rightarrow \lambda$ si es que aparece; y en ese caso, S no debe aparecer a la derecha.

Un posible pseudocódigo para este procedimiento es el siguiente:

Algoritmo para encontrar obtener una GLC propia

```

1:  $N \leftarrow$  conjunto de símbolos anulables
2: for each  $A \rightarrow \alpha \in P$  do
3:   if  $\alpha = X_1 X_2 \dots X_k$ , con  $X_{j_1}, X_{j_2}, \dots, X_{j_n} \in N$  then
4:      $P \leftarrow P \cup \{A \rightarrow \alpha' : \alpha' \text{ obtenida eliminando}$ 
       cada subconjunto en  $\{X_{j_1}, X_{j_2}, \dots, X_{j_n}\}$ ; si
       todos los  $X_1, X_2, \dots, X_n$  son anulables, no generar  $A \rightarrow \lambda\}$ 
5:   end if
6:   if  $\alpha = \lambda$  then
7:      $P \leftarrow P \setminus \{A \rightarrow \alpha\}$ 
8:   end if
9: end for

```

6.3.3. Reglas de renombramiento

Definición 6.14. Dada una gramática $G = \langle V_N, V_T, P, S \rangle$, una **regla de renombramiento** es una producción de la forma $A \rightarrow B \in P$, con $A, B \in V_N$.

Para eliminar este tipo de producciones, la idea es:

1. Construir, para cada símbolo no terminal A , el conjunto $N_A = \{B \in V_N \mid A \xRightarrow{*} B\}$.

2. Para cada $B \rightarrow \alpha$, agregar $A \rightarrow \alpha$ si $B \in N_A$.
3. Eliminar las producciones de la forma $A \rightarrow B$.

Algoritmo para eliminar reglas de renombramiento

```

1: for each  $A \in V_N$  do
2:    $N_A \leftarrow A$ 
3:   repeat
4:      $N_A \leftarrow N_A \cup \{C \in V_N \mid B \rightarrow C \in P \wedge B \in N_A\}$ 
5:   until  $N_A$  no cambió
6:   for each  $B \rightarrow \alpha \in P$  do
7:     if  $B \in N_A$  then
8:        $P \leftarrow P \cup \{A \rightarrow \alpha\}$ 
9:     end if
10:  end for
11:  for each  $A \rightarrow \alpha \in P$  do
12:    if  $\alpha \in V_N$  then
13:       $P \leftarrow P \setminus \{A \rightarrow \alpha\}$ 
14:    end if
15:  end for
16: end for

```

6.3.4. Factorización a izquierda

Si una gramática tiene producciones de la forma $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \dots \mid \alpha\beta_n \mid \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_k$, donde $\alpha \neq \lambda$ y ninguna de las γ_i empieza con α , entonces se puede introducir una ambigüedad cuando la cadena de entrada empieza con α (hay varias producciones que se pueden utilizar para derivar).

Para solucionar esto y eliminar dicha ambigüedad, se puede realizar una **factorización a izquierda**. Esto es, reemplazar las producciones mencionadas por:

$$\begin{aligned}
 A &\rightarrow \alpha A' \mid \gamma_1 \mid \gamma_2 \mid \dots \mid \gamma_k \\
 A' &\rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n
 \end{aligned}$$

, agregando el símbolo no terminal A' .

6.3.5. Recursividad a izquierda

Definición 6.15. Una gramática libre de contexto $G = \langle V_N, V_T, P, S \rangle$ es **recursiva a izquierda** si existe un símbolo no terminal $A \in V_N$ tal que para alguna expresión $\alpha \in (V_N \cup V_T)^*$, ocurre que $A \xRightarrow[L]{\neq} A\alpha$.

Se llama **recursión inmediata** cuando existe una producción $A \rightarrow A\alpha \in P$.

Ejemplo 6.3. La gramática $G = \langle \{E\}, \{+, *, (,), id\}, P, E \rangle$, con conjunto de producciones P formado por:

$$E \rightarrow E + E \mid E * E \mid (E) \mid id$$

presenta recursión inmediata a izquierda:

$$E \xRightarrow[L]{\neq} E + E$$

Ejemplo 6.4. La gramática $G = \langle \{A, B, C\}, \{a, b\}, P, A \rangle$, con conjunto de producciones P formado por:

$$\begin{aligned} A &\rightarrow BC \mid a \\ B &\rightarrow CA \mid Ab \end{aligned}$$

presenta recursión no inmediata a izquierda:

$$A \xRightarrow{L} BC \xRightarrow{L} AbC$$

La recursividad es problemática cuando se desea obtener un *parsing* para lenguajes libres de contexto con *una sola pasada*, con complejidad lineal (como en métodos que se verán más adelante). Por eso, es de interés poder eliminarla.

Teorema 6.1. *Todo lenguaje libre de contexto tiene una gramática no recursiva a izquierda que lo reconoce.*

Para demostrar este resultado, se construirá un algoritmo que transforma una GLC G recursiva a izquierda, en otra GLC G' no recursiva a izquierda y tal que $\mathcal{L}(G) = \mathcal{L}(G')$. Para eso, primero se define el concepto de ciclo:

Definición 6.16. Una gramática no tiene ciclos si para todo símbolo no terminal A , no existen derivaciones más a la izquierda $A \xRightarrow{L}^+ A$.

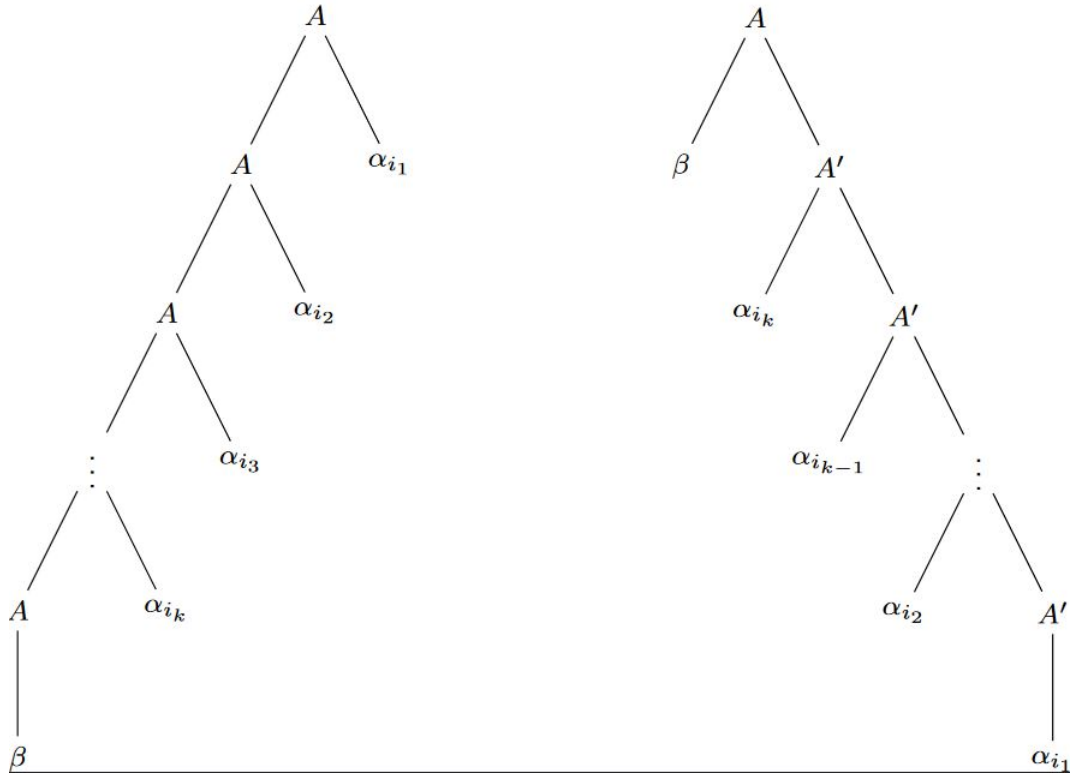


Figura 11: A la izquierda, un árbol de derivación para la cadena $\beta \alpha_{i_k} \alpha_{i_{k-1}} \dots \alpha_{i_2} \alpha_{i_1}$ en una gramática con producciones de la forma $A \rightarrow A \alpha_{i_1} \mid \dots \mid A \alpha_{i_m} \mid \beta$. A la derecha, un árbol de derivación resultante de haber eliminado la recursión a izquierda.

En las gramáticas libres de contexto, los ciclos ocurren cuando existen reglas borradoras o reglas de renombramiento. Utilizando los algoritmos mostrados antes, es posible eliminar estos ciclos.

La idea del algoritmo para eliminar la recursión a izquierda será hacer que si los árboles de derivación para una gramática *crecían hacia la izquierda* (como el que se ve a la izquierda en la Figura 11), conseguir una gramática alternativa, que genere el mismo lenguaje, pero cuyos árboles de derivación *crezcan hacia la derecha* (como el de la parte derecha de la Figura 11).

Lema 6.2. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto, cuyo conjunto de producciones P está dado por:

$$A \rightarrow A\alpha_1 \mid \dots \mid A\alpha_m \mid \beta_1 \mid \dots \mid \beta_n$$

, donde ninguno de los β_i empieza con el símbolo A (notar que G es recursiva a izquierda).

Sea $G' = \langle V_N \cup \{A'\}, V_T, P', S \rangle$, con A' un nuevo símbolo no terminal, y P' idéntico a P pero reemplazando las producciones que tienen A como cabeza por:

$$\begin{aligned} A &\rightarrow \beta_1 \mid \dots \mid \beta_n \mid \beta_1 A' \mid \dots \mid \beta_n A' \\ A' &\rightarrow \alpha_1 \mid \dots \mid \alpha_m \mid \alpha_1 A' \mid \dots \mid \alpha_m A' \end{aligned}$$

La gramática G' no es recursiva a izquierda, y $\mathcal{L}(G) = \mathcal{L}(G')$.

Demostración. ⁴ Notar que en la gramática G , las cadenas que se pueden obtener mediante una derivación más a la izquierda, con producciones que tengan como cabeza a A , están representadas por un conjunto regular que se puede expresar con la siguiente expresión regular:

$$(\beta_1 \mid \beta_2 \mid \dots \mid \beta_n)(\alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_m)^*$$

Esas cadenas son exactamente las mismas que las que se pueden obtener mediante una derivación más a la derecha usando una sola producción con cabeza A , y el resto con cabeza A' , en la gramática G' (la derivación resultante deja de ser más a la izquierda).

Todos los pasos de la derivación en G que no utilizan una producción con cabeza A se pueden hacer también en G' (ya que esas producciones se mantienen inalteradas). Luego, se concluye que $w \in \mathcal{L}(G)$, y que $\mathcal{L}(G') \subseteq \mathcal{L}(G)$.

Recíprocamente, se puede usar un razonamiento análogo para afirmar que: una derivación en G se supone más a la derecha, y se consideran secuencias de usar una única producción con cabeza A , y el resto con cabeza A' . En conclusión, $\mathcal{L}(G') = \mathcal{L}(G)$. \square

Observación. P' tiene el doble de producciones que P . Es decir, el algoritmo de eliminación de la recursión a izquierda duplica las producciones. En cuanto a la **complejidad**, esta es lineal en $|P|$.

Algoritmo de eliminación de la recursión inmediata a izquierda:

- Entrada: $G = \langle V_N, V_T, P, S \rangle$ gramática libre de contexto, sin producciones borradoras, símbolos inútiles ni ciclos.
- Salida: G' gramática libre de contexto, sin recursión inmediata a izquierda, tal que $\mathcal{L}(G) = \mathcal{L}(G')$.
- Procedimiento: sea $n = |V_N|$. Numerar los símbolos no terminales A_1, A_2, \dots, A_n , iterar sobre $i = 1, 2, \dots, n$, y en el paso i transformar las producciones con cabeza A_i , como en el Lema 6.2.

⁴Extraída de [A&Uv1] Lema 2.15 (p.154)

- Invariante del ciclo: todas las producciones con cabeza A_k , para $k = 1, \dots, i - 1$, fueron posiblemente transformadas a $A_k \rightarrow \alpha$, donde α empieza con un símbolo terminal, o un símbolo no terminal A_ℓ , con $\ell > k$.

Algoritmo de eliminación de la recursión no inmediata a izquierda⁵:

- Entrada: $G = \langle V_N, V_T, P, S \rangle$ gramática libre de contexto, sin producciones borradoras, símbolos inútiles ni ciclos.
- Salida: G' gramática libre de contexto, sin recursión a izquierda, tal que $\mathcal{L}(G) = \mathcal{L}(G')$.
- Procedimiento: sea $n = |V_N|$. Numerar los símbolos no terminales A_1, A_2, \dots, A_n . Modificar G hasta obtener G' donde para cada producción $A_k \rightarrow \alpha$, α empieza con un símbolo terminal, o un símbolo no terminal A_ℓ , con $\ell > k$.
- Complejidad: sea c la máxima cantidad de producciones con la misma cabeza. En el peor caso, se tiene:
 - Todas las producciones de G con cabeza A_1 tienen recursión inmediata a izquierda. Al transformarlas, pasan a ser $2c$ producciones con cabeza A_1 en G' .
 - Todas las producciones de G con cabeza A_2 tienen cuerpo que empieza con A_1 . Al transformarlas, pasan a ser $2 \times 2c \times c = (2c)^2$ producciones con cabeza A_2 en G' .
 - Todas las producciones de G con cabeza A_3 tienen cuerpo que empieza con A_1 . Al transformarlas, pasan a ser $2c \times 2c \times (2c)^2 = (2c)^4$ producciones con cabeza A_3 en G' .
 - Todas las producciones de G con cabeza A_4 tienen cuerpo que empieza con A_1 . Al transformarlas, pasan a ser $2c \times 2c \times (2c)^2 \times (2c)^4 = (2c)^8$ producciones con cabeza A_4 en G' .
 - ...
 - Todas las producciones de G con cabeza A_n tienen cuerpo que empieza con A_1 . Al transformarlas, pasan a ser

$$2c \times \prod_{i=0}^{n-1} (2c)^{2^i} = 2c \times (2c)^{2^0+2^1+\dots+2^{n-1}} = 2c \times (2c)^{2^n-1} = (2c)^{2^n}$$
 producciones con cabeza A_n en G' .
 - Por lo tanto, en peor caso G' tiene menos que $2 \times (2c)^{2^n}$ producciones.

Algoritmo para eliminar la recursión a izquierda

```

1: for  $i = 1, \dots, n$  do
2:   for  $j = 1, \dots, i - 1$  do
3:     if  $A_i \rightarrow A_j \gamma \wedge A_j \rightarrow \delta_1 \mid \delta_2 \mid \dots \mid \delta_m$  then
4:       Reemplazar  $A_i \rightarrow A_j \gamma$  por  $A_i \rightarrow \delta_1 \gamma \mid \delta_2 \gamma \mid \dots \mid \delta_m \gamma$ 
5:     end if
6:   end for
7:   Eliminar la recursión inmediata de las producciones con cabeza  $A_i$ 
8: end for

```

Observación. La performance de este algoritmo varía considerablemente con la elección del orden para la enumeración de los símbolos no terminales A_1, A_2, \dots, A_n .

⁵[A&Uv1] Algoritmo 2.13 (p.155)

6.3.6. Ordenando las modificaciones

Uno de los posibles órdenes correctos para realizar las modificaciones mencionadas en esta sección es:

1. Eliminar recursividad a izquierda.
2. Eliminar producciones λ .
3. Eliminar reglas de renombramiento.
4. Eliminar símbolos inactivos.
5. Eliminar símbolos inalcanzables.

6.3.7. Forma Normal de Chomsky

Definición 6.17. Sea L un lenguaje libre de contexto tal que $\lambda \notin L$. Existe una gramática $G = \langle V_N, V_T, P, S \rangle$ tal que $\mathcal{L}(G) = L$, y cuyas producciones son de la forma:

$$\begin{aligned} A &\rightarrow BC \\ A &\rightarrow a \end{aligned}$$

donde $a \in V_T$, y $A, B, C \in V_N$.

6.3.8. Forma Normal de Greibach

Definición 6.18. Sea L un lenguaje libre de contexto tal que $\lambda \notin L$. Existe una gramática $G = \langle V_N, V_T, P, S \rangle$ tal que $\mathcal{L}(G) = L$, y cuyas producciones son de la forma:

$$A \rightarrow a\alpha$$

donde $a \in V_T$, y $\alpha \in V_N^*$.

6.4. Lema de pumping para Lenguajes Libres de Contexto

Lema 6.3. (*Lema de pumping para lenguajes libres de contexto*) Si L es un lenguaje libre de contexto, entonces existe una longitud mínima n tal que: todas las cadenas $\alpha \in L$ con longitud mayor o igual que n pueden ser escritas de la forma $\alpha = rxyzs$, donde $|xyz| \leq n$, $|xz| > 0$, $\forall i \geq 0 : rx^i y z^i s \in L$. Formalmente:

$$\begin{aligned} L \text{ libre de contexto} &\Rightarrow \left(\exists n > 0 \mid \forall \alpha : (\alpha \in L \wedge |\alpha| \geq n) \Rightarrow \right. \\ &\left. \exists r, x, y, z, s \mid \left(\alpha = rxyzs \wedge |xyz| \leq n \wedge |xz| > 0 \wedge \forall i \geq 0 : rx^i y z^i s \in L \right) \right) \end{aligned}$$

Demostración. Sea G una gramática libre de contexto tal que $L = \mathcal{L}(G)$.

Sea $a = \max \{ |\beta| : A \rightarrow \beta \in P \}$.

Sea $n = a^{|V_N|+1}$; considerar la cadena $\alpha \in L$ tal que $|\alpha| \geq n$. Sea $\mathcal{T}(S)$ un árbol de derivación de altura mínima para la cadena α . Por el Lema 6.1, resulta que $a^h \geq |\alpha|$, por lo que:

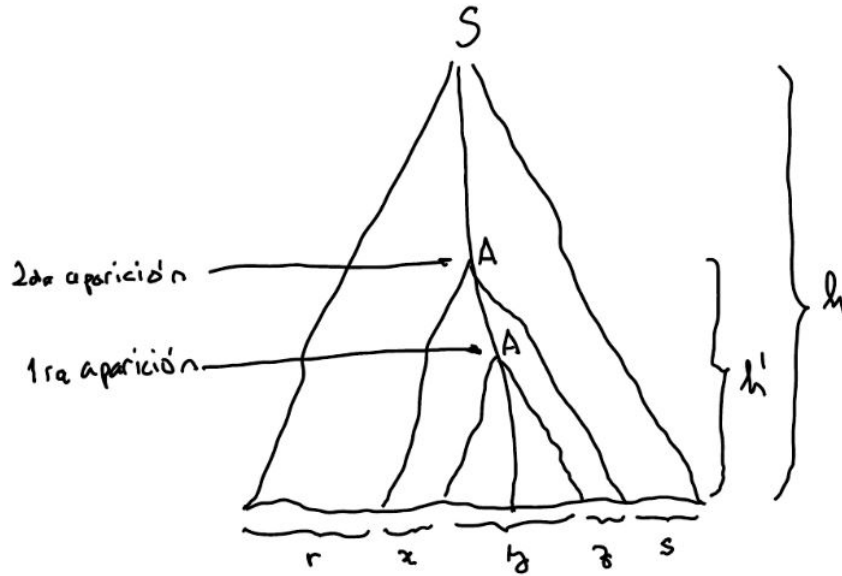
$$a^h \geq |\alpha| \geq n = a^{|V_N|+1}$$

Si $a < 2$, entonces $a = 0$ o $a = 1$; en ambos casos, las cadenas pertenecientes al lenguaje L tienen longitudes no mayores que 1. Luego, eligiendo $n \geq 2$ el lema es trivialmente verdadero.

Si $a > 2$, entonces:

$$h \geq |V_N| + 1$$

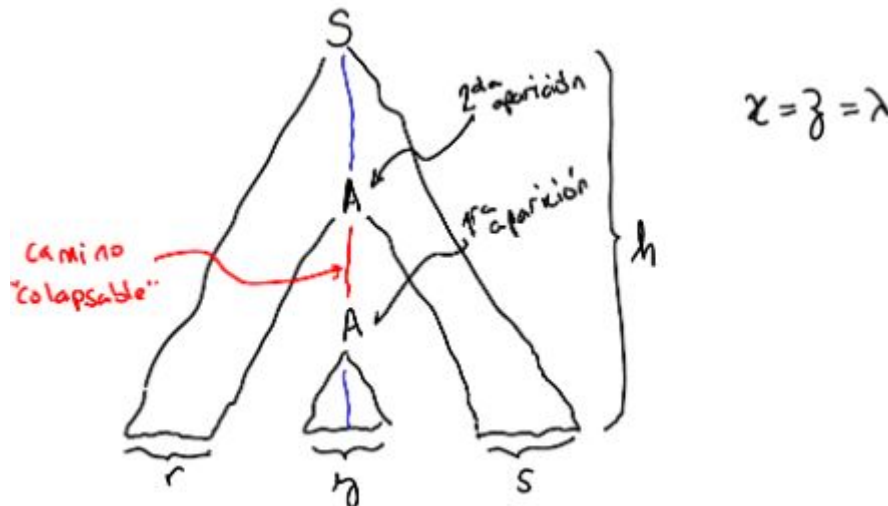
Entonces, existirá algún símbolo en α tal que su camino en $\mathcal{T}(S)$ será de longitud $h \geq |V_N| + 1$. Como la cantidad de símbolos no terminales es $|V_N|$, entonces en ese camino debe existir un símbolo no terminal repetido: sea A ese símbolo. Recorriendo el camino en forma ascendente, considerar la primera y la segunda aparición del símbolo repetido A :



Como se puede ver en la imagen, la segunda aparición de A da lugar a la cadena xyz , y como hay garantía de que esa segunda aparición de A está a una distancia $h' \leq |V_N| + 1$ respecto de la base, entonces vale que:

$$|xyz| \leq a^{h'} \leq a^{|V_N|+1} = n$$

Cuando las cadenas x, z son ambas nulas, el camino de la segunda aparición de A hasta la primera aparición puede ser *colapsado*, porque al hacer eso la cadena generada por el árbol resultante será la misma, rys :



Como el árbol considerado se tomó de altura mínima entre todos los que sirven para derivar la cadena α , entonces siempre se puede encontrar un camino de longitud máxima entre la raíz y alguna hoja en el que x, z no sean simultáneamente nulas. Esto es así porque, si no lo fuera, todos los caminos que van desde la raíz S hasta una hoja (que sean de longitud máxima, igual a la altura h del árbol) podrían ser *colapsados* como se mostró recién; y entonces, se podría obtener un árbol de derivación para α con menor altura, lo cual contradice el hecho de que el árbol era de altura mínima.

Finalmente, como $S \xRightarrow{*} rAs$, y $A \xRightarrow{*} y$, entonces es cierto que $S \xRightarrow{*} rAs \xRightarrow{*} rys$, con lo cual $rys = rx^0yz^0s \in L$, lo cual demuestra que vale el caso base del resultado.

Teniendo en cuenta la hipótesis inductiva:

$$S \xRightarrow{*} rx^{i-1}Az^{i-1}s$$

, y que $A \xRightarrow{*} xyz$, entonces es cierto que:

$$S \xRightarrow{*} rx^{i-1}Az^{i-1}s \xRightarrow{*} rx^{i-1}xyz^{i-1}s = rx^i y z^i s$$

Por lo tanto, $rx^i y z^i s \in L$. □

6.5. Propiedades de Lenguajes Libres de Contexto

6.5.1. Unión de lenguajes libres de contexto

Teorema 6.2. *Si L_1 y L_2 son dos lenguajes libres de contexto, entonces $L_1 \cup L_2$ también es libre de contexto.*

Demostración. Como L_1 y L_2 son libres de contexto, entonces existen GLCs $G_1 = \langle V_{N_1}, V_T, P_1, S_1 \rangle$ y $G_2 = \langle V_{N_2}, V_T, P_2, S_2 \rangle$ tales que $\mathcal{L}(G_1) = L_1$, y $\mathcal{L}(G_2) = L_2$.

Sin pérdida de generalidad, suponer que $V_{N_1} \cap V_{N_2} = \emptyset$ (si no fuera así, se puede renombrar los símbolos para que sí lo sea). Sea G la siguiente gramática:

$$G = \langle V_{N_1} \cup V_{N_2} \cup \{S\}, V_T, P_1 \cup P_2 \cup \{S \rightarrow S_1, S \rightarrow S_2\}, S \rangle$$

Es fácil ver que $\forall \alpha \in V_T^*, \alpha \in \mathcal{L}(G) \Leftrightarrow \alpha \in \mathcal{L}(G_1) \cup \mathcal{L}(G_2)$. Y como G es libre de contexto (G_1 y G_2 lo eran, y las dos producciones que se agregaron cumplen con la restricción de las GLCs), entonces $\mathcal{L}(G) = L_1 \cup L_2$ es un lenguaje libre de contexto. □

6.5.2. Concatenación de lenguajes libres de contexto

Teorema 6.3. *Si L_1 y L_2 son dos lenguajes libres de contexto, entonces $L_1 L_2$ también es libre de contexto.*

Demostración. Como L_1 y L_2 son libres de contexto, entonces existen GLCs $G_1 = \langle V_{N_1}, V_T, P_1, S_1 \rangle$ y $G_2 = \langle V_{N_2}, V_T, P_2, S_2 \rangle$ tales que $\mathcal{L}(G_1) = L_1$, y $\mathcal{L}(G_2) = L_2$.

Sin pérdida de generalidad, suponer que $V_{N_1} \cap V_{N_2} = \emptyset$ (si no fuera así, se puede renombrar los símbolos para que sí lo sea). Sea G la siguiente gramática:

$$G = \langle V_{N_1} \cup V_{N_2} \cup \{S\}, V_T, P_1 \cup P_2 \cup \{S \rightarrow S_1 S_2\}, S \rangle$$

Es fácil ver que $\forall \alpha \in V_T^*, \alpha \in \mathcal{L}(G) \Leftrightarrow \alpha \in \mathcal{L}(G_1)\mathcal{L}(G_2)$. Y como G es libre de contexto (G_1 y G_2 lo eran, y la producción que se agregó cumple con la restricción de las GLCs), entonces $\mathcal{L}(G) = L_1L_2$ es un lenguaje libre de contexto. \square

6.5.3. Clausura positiva de un lenguaje libre de contexto

Teorema 6.4. *Si L es un lenguaje libre de contexto, entonces L^+ también es libre de contexto.*

Demostración. Como L es libre de contexto, entonces existe una GLC $G = \langle V_N, V_T, P, S \rangle$ tal que $\mathcal{L}(G) = L$.

Sea G' la siguiente gramática:

$$G' = \langle V_N \cup \{S'\}, V_T, P \cup \{S' \rightarrow SS', S' \rightarrow S\}, S' \rangle$$

Es fácil ver que $\forall \alpha \in V_T^*, \alpha \in \mathcal{L}(G') \Leftrightarrow \alpha \in \mathcal{L}(G)^+$. Y como G' es libre de contexto (G lo era, y las producciones que se agregaron cumplen con la restricción de las GLCs), entonces $\mathcal{L}(G') = L^+$ es un lenguaje libre de contexto. \square

6.5.4. Clausura de Kleene de un lenguaje libre de contexto

Teorema 6.5. *Si L es un lenguaje libre de contexto, entonces L^* también es libre de contexto.*

Demostración. Como L es libre de contexto, entonces existe una GLC $G = \langle V_N, V_T, P, S \rangle$ tal que $\mathcal{L}(G) = L$.

Sea G' la siguiente gramática:

$$G' = \langle V_N \cup \{S'\}, V_T, P \cup \{S' \rightarrow SS', S' \rightarrow \lambda\}, S' \rangle$$

Es fácil ver que $\forall \alpha \in V_T^*, \alpha \in \mathcal{L}(G') \Leftrightarrow \alpha \in \mathcal{L}(G)^*$. Y como G' es libre de contexto (G lo era, y las producciones que se agregaron cumplen con la restricción de las GLCs), entonces $\mathcal{L}(G') = L^*$ es un lenguaje libre de contexto. \square

6.5.5. Intersección de lenguajes libres de contexto

Observación. Si L_1 y L_2 son dos lenguajes libres de contexto, entonces $L_1 \cap L_2$ **no necesariamente** es libre de contexto.

Considerar el siguiente ejemplo:

$$L_1 = \{a^n b^m c^l : n, m, l \geq 0 \wedge n = m\}$$

$$L_2 = \{a^n b^m c^l : n, m, l \geq 0 \wedge m = l\}$$

La gramática $G_1 = \langle \{S, A, C\}, \{a, b, c\}, \{S \rightarrow AC, S \rightarrow C, A \rightarrow aAb, A \rightarrow ab, C \rightarrow cC, C \rightarrow \lambda\}, S \rangle$ es libre de contexto, y genera L_1 . Se puede hacer un procedimiento análogo para L_2 , y concluir que tanto L_1 como L_2 son lenguajes libres de contexto.

Considerar ahora su intersección:

$$L_1 \cap L_2 = \{a^n b^m c^l : n, m, l \geq 0 \wedge n = m = l\}$$

Este lenguaje no es libre de contexto (se puede probar usando el contrarrecíproco del lema de pumping para lenguajes libres de contexto).

6.6. Lenguajes Libres de Contexto Determinísticos

Definición 6.19. Un lenguaje L es libre de contexto determinístico (LICD) si existe una autómatas de pila determinístico (APD) M tal que $L = \mathcal{L}(M)$.

El conjunto de los lenguajes libres de contexto determinísticos es **cerrado por el complemento**. La manera *ingenua* de obtener el APD *complemento* para un APD dado es hacer que los estados finales pasen a ser no-finales, y viceversa (al igual que se hace con los autómatas finitos).

Sin embargo, esta solución tiene problemas:

- Puede ocurrir que el autómatas original no consuma totalmente todas las cadenas de entrada (puede alcanzar una configuración desde la cual ninguna transición es posible, o entrar en un ciclo infinito de transiciones λ). En este caso, la cadena no consumida es rechazada antes y después de intercambiar los estados finales con los no-finales, no cumpliendo entonces que el supuesto *autómatas complemento* reconozca el lenguaje complemento.
- Puede ocurrir también que el autómatas consuma todas las cadenas de entrada, pero que exista alguna cadena tal que, después de haber sido consumida, haga que el autómatas quede en una configuración tal que sea posible realizar transiciones λ pasando por estados finales y no-finales. Entonces, al pasar al supuesto *autómatas complemento*, ocurre lo mismo que en el caso anterior.

La no-trivialidad de la solución motiva los siguientes resultados.

Teorema 6.6. \dagger Dado un autómatas de pila determinístico $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$, existe un autómatas de pila determinístico M' equivalente, que siempre consuma la cadena de entrada.

Demostración. Sea $M' = \langle Q \cup \{q'_0, d, f\}, \Sigma, \Gamma \cup \{X_0\}, \delta', q'_0, X_0, F \cup \{f\} \rangle$ un APD, donde:

1. $\delta'(q'_0, \lambda, X_0) = (q_0, Z_0 X_0)$, para evitar que el APD se detenga por haberse vaciado la pila.
2. $\forall q \in Q, a \in \Sigma, Z \in \Gamma : \delta(q, a, Z) = \delta(q, \lambda, Z) = \emptyset \Rightarrow \delta'(q, a, Z) = (d, Z)$, y además, $\forall q \in Q, a \in \Sigma : \delta'(q, a, X_0) = (d, X_0)$.
3. $\forall a \in \Sigma, Z \in \Gamma \cup \{X_0\} : \delta'(d, a, Z) = (d, Z)$, para completar el autómatas con un *estado trampa*.
4. Si para $q \in Q, Z \in \Gamma$ vale que: $\forall i \geq 1 : \exists q_i, \gamma_i$ para los cuales $(q, \lambda, Z) \stackrel{i}{\vdash} (q_i, \lambda, \gamma_i)$, entonces:
 - si ningún q_i es un estado final, entonces $\delta'(q, \lambda, Z) = (d, Z)$.
 - si alguno sí lo es, entonces $\delta'(q, \lambda, Z) = (f, Z)$.
5. $\forall Z \in \Gamma \cup \{X_0\} : \delta'(f, \lambda, Z) = (d, Z)$.
6. $\forall q \in Q, a \in \Sigma \cup \{\lambda\}, Z \in \Gamma : \text{si } \delta'(q, a, Z) \text{ no ha sido definida por las reglas 2 o 4, entonces } \delta'(q, a, Z) = \delta(q, a, Z)$.

Primero se verá que el APD M' consume siempre toda la cadena de entrada. Suponer que esto no ocurre, es decir que existe una cadena de entrada xy tal que M' no consume más allá del prefijo x . Entonces, debe ocurrir que

$$(q'_0, xy, X_0) \xrightarrow{*}_M (q, y, Z_1 \dots Z_k X_0)$$

, y que a partir de esa configuración instantánea, no se consuma más entrada. Sin embargo, esto no puede pasar porque:

- no pasa que no existan transiciones posibles (por la regla 2), y
- tampoco pasa que se haya entrado en un ciclo infinito de transiciones λ (por la regla 4).

Por la regla 4, en algún momento se debe consumir Z_1 , y lo mismo sucede para los demás Z_i de la pila, hasta llegar a X_0 . En ese momento, tomando $y = ay'$ se tiene que, por la regla 2:

$$\delta'(q, ay', X_0) \vdash \delta'(q, y', X_0)$$

□

Teorema 6.7. † *El complemento de un lenguaje libre de contexto determinístico es un lenguaje libre de contexto determinístico.*

Demostración. Dado el APD $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$, se puede afirmar, sin pérdida de generalidad (por el Teorema 6.6) que M consume toda la cadena de entrada.

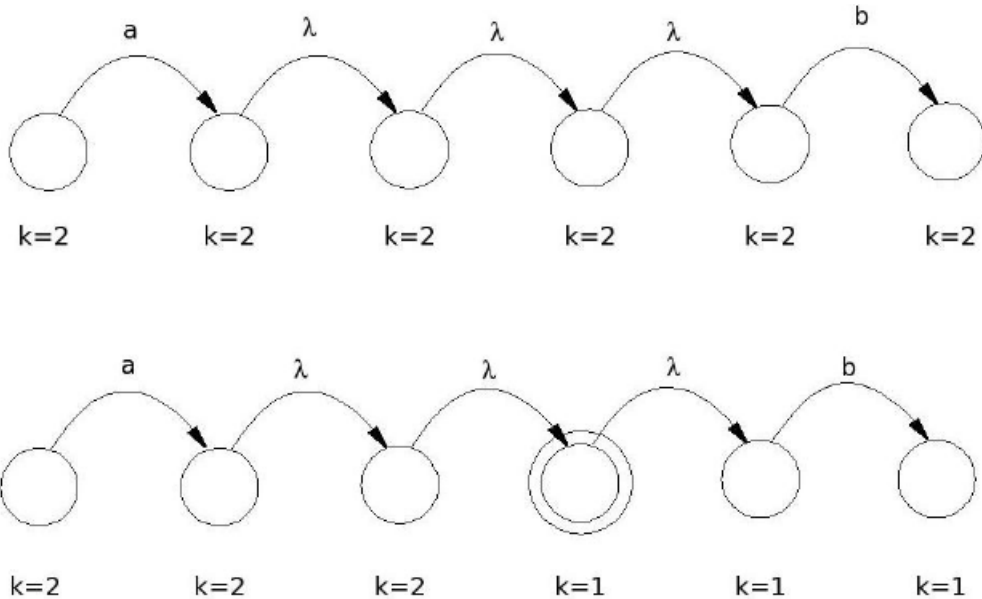
Sea $M' = \langle Q', \Sigma, \Gamma, \delta', q'_0, Z_0, F' \rangle$, donde:

$$Q' = \{[q, k] : q \in Q \wedge k \in \{1, 2, 3\}\}$$

$$F' = \{[q, 3] : q \in Q\}$$

$$q'_0 = \begin{cases} [q_0, 1] & \text{si } q_0 \in F \\ [q_0, 2] & \text{si } q_0 \notin F \end{cases}$$

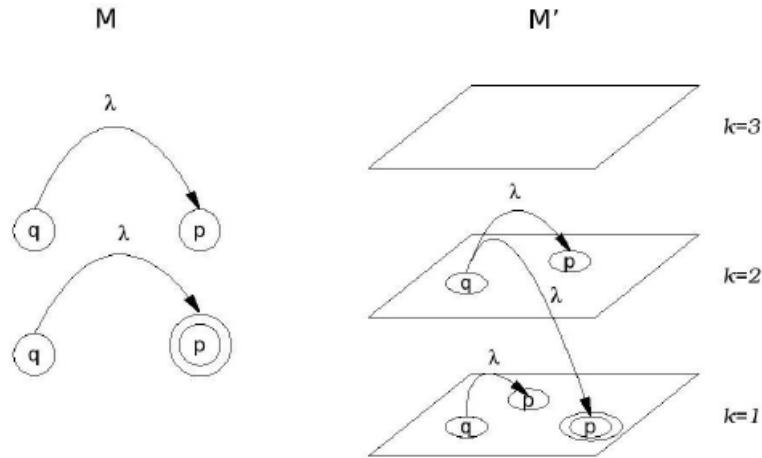
El propósito de k es detectar si entre transiciones con consumo de entrada se pasó o no por un estado final. En caso de que se haya pasado por un estado final, entonces se hará $k = 1$; si no, será $k = 2$.



En cuanto a la función de transición δ' , queda definida por:

1. Si $\delta(q, \lambda, Z) = (p, \gamma)$ entonces, para $k = 1$ o $k = 2$:

$$\delta'([q, k], \lambda, Z) = \begin{cases} ([p, 1], \gamma) & \text{si } k = 1 \vee p \in F \\ ([p, 2], \gamma) & \text{en caso contrario} \end{cases}$$

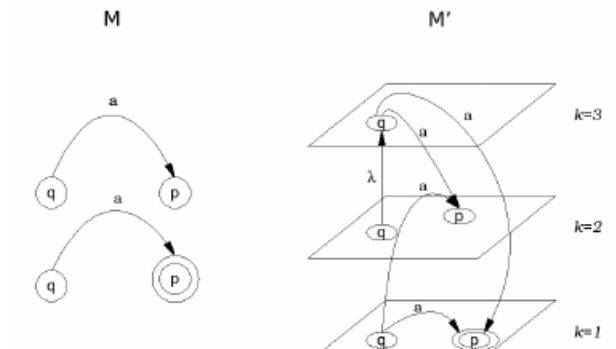


2. Si $\delta(q, a, Z) = (p, \gamma)$ entonces, para $k = 1$ o $k = 2$:

$$\delta'([q, 2], \lambda, Z) = ([q, 3], Z)$$

, y por otro lado:

$$\delta'([q, 1], a, Z) = \delta'([q, 3], a, Z) = \begin{cases} ([p, 1], \gamma) & \text{si } p \in F \\ ([p, 2], \gamma) & \text{si } p \notin F \end{cases}$$



□

6.7. Lenguajes Libres de Contexto No Determinísticos

Teorema 6.8. *Existe un lenguaje libre de contexto que es no determinístico.*

Demostración. Considerar el lenguaje $L = \{a^k b^k c^k : k \geq 0\}$. Se vio anteriormente que L no es libre de contexto.

Considerar ahora el complemento de L . Como se puede ver a continuación, este lenguaje se puede escribir como la unión de dos lenguaje libres de contexto y un lenguaje regular.

$$\bar{L} = \{a^i b^j c^k : i \neq j \wedge i, j, k \geq 0\} \cup \{a^i b^j c^k : j \neq k \wedge i, j, k \geq 0\} \cup \overline{a^* b^* c^*}$$

Por lo tanto, \bar{L} es un lenguaje libre de contexto. Por el Teorema 6.7, es cierto que el complemento de un lenguaje libre de contexto determinístico (LICD) es también libre de contexto determinístico. Por lo tanto, si \bar{L} fuera LICD, entonces su complemento $\overline{\bar{L}}$ también sería LICD. Pero $\overline{\bar{L}} = L$, que no es ni siquiera libre de contexto. Por lo tanto, \bar{L} debe ser un lenguaje libre de contexto no determinístico (LICND). \square

6.8. Fuentes

- Julio Jacobo. Teoría de Lenguajes, Clase Teórica 6. Primer cuatrimestre, 2022.
- Verónica Becher. Teoría de Lenguajes, Clase Teórica 8. Primer cuatrimestre, 2022.
- Manuel Panichelli. Teoría de Lenguajes, Clase Práctica 8. Primer cuatrimestre, 2022.
- Ariel Arbiser. Teoría de Lenguajes, Clase Práctica 9. Primer cuatrimestre, 2022.

7. Lenguajes y Parsing LL

Dentro de las gramáticas libres de contexto (GLC), existen gramáticas que generan lenguajes que admiten un *parsing* en tiempo lineal en el tamaño de entrada, leyendo de izquierda a derecha con *una sola pasada* por la entrada. Estas son las gramáticas *LL* y *LR*.

Una gramática se dice *LL* si es *LL(k)* para algún $k \geq 1$ entero. Estas son gramáticas libres de contexto para las cuales la derivación más a la izquierda queda completamente determinada por los símbolos ya leídos de la entrada, más k símbolos adicionales (llamados *lookahead*).

Estas gramáticas admiten un *parsing top-down*; esto es, encontrar las producciones que forman la derivación desde el símbolo inicial hasta la cadena, una por una. Cada paso de la derivación se resuelve en tiempo constante, con lo cual el tiempo total es lineal en el tamaño de la entrada.

Además, todo lenguaje *LL(k)* tiene un autómata de pila determinístico, con un solo estado, que lo reconoce.

Definición 7.1. Sea $G = \langle V_N, V_T, P, S \rangle$ una **GLC no ambigua**, y sea $w = a_1a_2\dots a_n \in \mathcal{L}(G)$. Entonces hay una única secuencia de formas sentenciales $\alpha_0, \alpha_1, \dots, \alpha_m$ tal que $\alpha_i \xRightarrow{L} \alpha_{i+1}$, para $i = 0, 1, \dots, m-1$, con $\alpha_0 = S$ y $\alpha_m = w$.

El *parsing* a izquierda de w es la secuencia de las m producciones usadas en la derivación de w .

Las gramáticas *LL(k)* cumplen que si $\alpha_i = a_1\dots a_j A \beta$, entonces α_{i+1} es determinable conociendo únicamente $a_1\dots a_j$ y k símbolos más de la entrada $a_{j+1}\dots a_{j+k}$.

Ejemplo 7.1. $G = \langle \{S, A\}, \{a, b, c, d\}, P, S \rangle$, con producciones P dadas por:

$$\begin{aligned} S &\rightarrow cAd \\ A &\rightarrow ab \mid a \end{aligned}$$

Considerar la cadena $w = cad$:

$$\begin{aligned} S &\xRightarrow{L} cAd \xRightarrow{L} cabd \\ S &\xRightarrow{L} cAd \xRightarrow{L} cad \end{aligned}$$

Los dos símbolos que se leen de la entrada, junto con el símbolo no terminal de más a la izquierda, determinan qué producción usar. Luego, G es *LL(2)*.

Ejemplo 7.2. $G = \langle \{S, A\}, \{a, b\}, P, S \rangle$, con producciones P dadas por:

$$\begin{aligned} S &\rightarrow aAS \mid b \\ A &\rightarrow a \mid bSA \end{aligned}$$

Considerar la cadena $w = aaaab$:

$$S \xRightarrow{L} aAS \xRightarrow{L} aaS \xRightarrow{L} aaaAS \xRightarrow{L} aaaaS \xRightarrow{L} aaaab$$

En este caso, el símbolo que se lee de la entrada, junto con el símbolo no terminal de más a la izquierda, determinan qué producción usar. Luego, G es *LL(1)*.

Ejemplo 7.3. $G = \langle \{S, A, B\}, \{a, b, 0, 1\}, P, S \rangle$, con producciones P dadas por:

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow aAb \mid 0 \\ B &\rightarrow aBbb \mid 1 \end{aligned}$$

Esta gramática genera el lenguaje $\mathcal{L}(G) = \{a^n 0 b^n : n \geq 0\} \cup \{a^n 1 b^{2n} : n \geq 0\}$. Notar que para todo $k \geq 1$,

$$\begin{aligned} S &\xRightarrow[L]{*} A \xRightarrow[L]{*} a^k 0 b^k \\ S &\xRightarrow[L]{*} B \xRightarrow[L]{*} a^k 1 b^{2k} \end{aligned}$$

Los primeros k símbolos en $a^k 0 b^k$ y $a^k 1 b^{2k}$ coinciden, por lo que no son suficientes para determinar qué producción usar. Luego, G no es $LL(k)$ para ningún k .

7.1. Primeros de una cadena

Los símbolos directrices son la noción central para poder realizar un *parsing* para una gramática $LL(k)$. Para definirlos, primero se debe definir los conceptos de *primeros* y *siguientes*.

Definición 7.2. Sean $k \geq 1$, $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto, y $\alpha \in (V_N \cup V_T)^*$ una cadena. Se define $\text{PRIMEROS}_k : (V_N \cup V_T)^* \rightarrow \mathcal{P}(V_T)$ de α como:

$$\text{PRIMEROS}_k(\alpha) = \left\{ z \in V_T^+ : \left(|z| < k \wedge \alpha \xRightarrow[L]{*} z \right) \vee \left(|z| = k \wedge \alpha \xRightarrow[L]{*} zw, \text{ para alguna } w \in V_T^* \right) \right\}$$

En el caso particular de $k = 1$:

$$\text{PRIMEROS}(\alpha) = \{ t \in V_T : \alpha \xRightarrow[L]{*} tw, \text{ para alguna } w \in V_T^* \}$$

Intuitivamente, se trata del conjunto que contiene los símbolos terminales con los que pueden empezar las cadenas que se derivan de α .

Esto da lugar a la siguiente definición para gramáticas LL :

Definición 7.3. Una gramática libre de contexto $G = \langle V_N, V_T, P, S \rangle$ es $LL(k)$ cuando ocurre que: cada vez que

$$\begin{aligned} S \xRightarrow[L]{*} wA\alpha &\Rightarrow_L w\beta\alpha \xRightarrow[L]{*} wx \\ S \xRightarrow[L]{*} wA\alpha &\Rightarrow_L w\gamma\alpha \xRightarrow[L]{*} wy \\ \text{PRIMEROS}_k(x) &= \text{PRIMEROS}_k(y) \end{aligned}$$

, entonces se cumple que $\beta = \gamma$.

También surge el siguiente resultado para decidir si una gramática es $LL(k)$:

Teorema 7.1. Una gramática libre de contexto $G = \langle V_N, V_T, P, S \rangle$ es $LL(k)$ si, y sólo si, para todos los $wA\alpha$ tales que $S \xRightarrow[L]{*} wA\alpha$, y para todo par de producciones $A \rightarrow \beta$ y $A \rightarrow \gamma$, con $\beta \neq \gamma$, vale que $\text{PRIMEROS}_k(\beta\alpha) \cap \text{PRIMEROS}_k(\gamma\alpha) = \emptyset$.

Demostración. Probando las dos implicaciones por separado:

- \Rightarrow) Por el absurdo. Suponer que $S \xRightarrow{*}_L wA\alpha$, y que existen producciones $A \rightarrow \beta$ y $A \rightarrow \gamma$, con $\beta \neq \gamma$, y

$$S \xRightarrow{*}_L wA\alpha \Rightarrow_L w\beta\alpha \xRightarrow{*}_L wxy$$

$$S \xRightarrow{*}_L wA\alpha \Rightarrow_L w\gamma\alpha \xRightarrow{*}_L wxz$$

, con $x \in \text{PRIMEROS}_k(\beta\alpha) \cap \text{PRIMEROS}_k(\gamma\alpha)$ (con lo cual estos conjuntos no son disjuntos), tales que si $|x| < k$ entonces $y = z = \lambda$.

Como $\beta \neq \gamma$, entonces por la definición 7.3, G no es $LL(k)$. Sin embargo, esto contradice la hipótesis. La contradicción proviene de haber supuesto que $\text{PRIMEROS}_k(\beta\alpha) \cap \text{PRIMEROS}_k(\gamma\alpha) \neq \emptyset$.

- \Leftarrow) Por el absurdo. Suponer que G no es $LL(k)$. Luego, por la definición 7.3, existen dos derivaciones

$$S \xRightarrow{*}_L wA\alpha \Rightarrow_L w\beta\alpha \xRightarrow{*}_L wx$$

$$S \xRightarrow{*}_L wA\alpha \Rightarrow_L w\gamma\alpha \xRightarrow{*}_L wy$$

, y existe $z \in V_T^*$ con $|z| \leq k$, tal que $\{z\} = \text{PRIMEROS}_k(x) = \text{PRIMEROS}_k(y)$.

Entonces, $z \in \text{PRIMEROS}_k(\beta\alpha)$, y $z \in \text{PRIMEROS}_k(\gamma\alpha)$. Con lo cual, los conjuntos no son disjuntos. Esto es una contradicción de la hipótesis, proveniente de suponer que G no era $LL(k)$.

□

Ejemplo 7.4. $G = \langle \{S, A\}, \{a, c\}, P, S \rangle$, con producciones dadas por:

$$S \rightarrow cAa$$

$$A \rightarrow \lambda \mid a$$

Tomando la cadena $w = caa$, se tiene que $S \xRightarrow{*}_L cAa$.

Por un lado, $\text{PRIMEROS}(a) = \{a\}$ y $\text{PRIMEROS}(\lambda) = \{\}$ son disjuntos.

Sin embargo, por otro lado, $\text{PRIMEROS}(\lambda a) = \{a\}$, y $\text{PRIMEROS}(aa) = \{a\}$ no lo son.

Luego, G no es $LL(1)$.

7.2. Siguietes de un símbolo no terminal

Definición 7.4. Sean $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto, y $A \in V_N$ un símbolo no terminal. Se define $\text{SIGUIENTES} : V_N \rightarrow \mathcal{P}(V_T)$ de A como:

$$\text{SIGUIENTES}(A) = \{z \in V_T \cup \{\lambda\} : S \xRightarrow{*} \alpha A \gamma, z \in \text{PRIMEROS}(\gamma)\}$$

Alternativamente, se puede definir como:

$$\text{SIGUIENTES}(A) = \{z \in V_T \cup \{\lambda\} : S \xRightarrow{*} \alpha A z \beta\}$$

Observación. Esta definición es la que se estudió en las clases teóricas. En las prácticas, se utilizó una versión en la que λ no es un valor posible para SIGUIENTES .

Intuitivamente, los siguientes de un símbolo no terminal A conforman el conjunto de símbolos terminales que pueden aparecer inmediatamente *después* de A en una derivación.

Para calcularlo, se puede iterar mientras los conjuntos de siguientes cambien, haciendo:

- Si una producción es de la forma $B \rightarrow \alpha A \gamma$, agregar $\text{PRIMEROS}(\gamma)$ a $\text{SIGUIENTES}(A)$. Si además γ es anulable, entonces agregar $\text{SIGUIENTES}(B)$ a $\text{SIGUIENTES}(A)$.

Corolario. (del Teorema 7.1) Una gramática libre de contexto $G = \langle V_N, V_T, P, S \rangle$ es $LL(1)$ si, y sólo si, para todos los $wA\alpha$ tales que $S \xRightarrow{*}_L wA\alpha$ y para todo par de producciones $A \rightarrow \beta$ y $A \rightarrow \gamma$, con $\beta \neq \gamma$, vale que:

- $\text{PRIMEROS}_1(\beta) \cap \text{PRIMEROS}_1(\gamma) = \emptyset$, si β, γ no son anulables.
- $(\text{PRIMEROS}_1(\beta) \cup \text{SIGUIENTES}(A)) \cap \text{PRIMEROS}_1(\gamma) = \emptyset$, si β es anulable y γ no.
- $\text{PRIMEROS}_1(\beta) \cap (\text{PRIMEROS}_1(\gamma) \cup \text{SIGUIENTES}(A)) = \emptyset$, si γ es anulable y β no.
- $(\text{PRIMEROS}_1(\beta) \cup \text{SIGUIENTES}(A)) \cap (\text{PRIMEROS}_1(\gamma) \cup \text{SIGUIENTES}(A)) = \emptyset$, si β, γ son anulables.

7.3. Símbolos Directrices

Definición 7.5. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto, y sean $A \in V_N, \beta \in (V_N \cup V_T)^*$. Se define la función de símbolos directrices $SD : P \rightarrow \mathcal{P}(V_T)$ como

$$SD(A \rightarrow \beta) = \begin{cases} \text{PRIMEROS}(\beta) & \text{si } \beta \text{ no es anulable} \\ \text{PRIMEROS}(\beta) \cup \text{SIGUIENTES}(A) & \text{si } \beta \text{ es anulable} \end{cases}$$

Observación. Una gramática libre de contexto $G = \langle V_N, V_T, P, S \rangle$ es $LL(1)$ si, y sólo si, para cada símbolo no terminal $A \in V_N$ y cada símbolo terminal $a \in V_T$, hay a lo sumo una única producción $A \rightarrow \beta$ tal que $a \in SD(A \rightarrow \beta)$.

Es decir: una gramática libre de contexto es $LL(1)$ si, y sólo si, la intersección de los símbolos directrices es nula, para cada subconjunto de sus producciones que compartan cabeza.

Esto da un **método eficiente** para determinar si una gramática es $LL(1)$ o no, y es la base para el algoritmo de *parsing* $LL(k)$.

7.4. Ambigüedad

Teorema 7.2. Toda gramática $LL(k)$ es **no ambigua**.

Demostración. Sea G una gramática $LL(k)$. Suponer que G es ambigua. Entonces, existe una cadena w con dos derivaciones distintas:

$$\begin{aligned} S &\xRightarrow{*}_L \alpha_1 \xRightarrow{*}_L \alpha_2 \dots \xRightarrow{*}_L \alpha_n \xRightarrow{*}_L w \\ S &\xRightarrow{*}_L \beta_1 \xRightarrow{*}_L \beta_2 \dots \xRightarrow{*}_L \beta_m \xRightarrow{*}_L w \end{aligned}$$

Sea i el índice mínimo tal que $\alpha_i \neq \beta_i$. Luego, $\alpha_{i-1} = \beta_{i-1}$, por lo que el símbolo no terminal más a la izquierda es el mismo en α_{i-1} y en β_{i-1} . Entonces:

$$\begin{array}{ccc} S \xRightarrow{*}_L \alpha_{i-1} & \xRightarrow{*}_L & \alpha_i \xRightarrow{*}_L w \\ S \xRightarrow{*}_L \beta_{i-1} & \xRightarrow{*}_L & \beta_i \xRightarrow{*}_L w \\ w & = & w \end{array}$$

Sin embargo, se cumple $\alpha_i \neq \beta_i$ por cómo se eligió i . Por la definición 7.3, la gramática G no es $LL(k)$. Esto contradice lo que se supuso al principio. Por lo tanto, G es $LL(k)$. \square

7.5. Recursividad a izquierda

Teorema 7.3. *Toda gramática $LL(k)$ sin ciclos y sin símbolos inútiles es no recursiva a izquierda.*

Demostración. Sean $k \geq 1$ fijo, $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto. Se extiende a G con el símbolo no terminal S_k y el símbolo terminal \perp , tales que $S_k \rightarrow S\perp^k \in P$.

Suponer que G es recursiva a izquierda. Luego, existen reglas de la forma $A \rightarrow \gamma$, $A \rightarrow B\beta$, con $\gamma \neq B\beta$, y $B\beta \xRightarrow{*} A\alpha$, donde $\lambda \notin \text{PRIMEROS}(\alpha)$. Por lo tanto, $A \xRightarrow{*} A\alpha$.

Entonces,

$$\begin{array}{lcl} S_k \xRightarrow{*} xA... \Rightarrow xB\beta... \xRightarrow{*} xA\alpha^k... & \Rightarrow & x\gamma\alpha^k... \xRightarrow{*} xz... \\ S_k \xRightarrow{*} xA... \Rightarrow xB\beta... \xRightarrow{*} xA\alpha^k... & \Rightarrow & xB\beta\alpha^k... \xRightarrow{*} xz... \end{array}$$

(esto último es porque $x B\beta \alpha^k \dots \Rightarrow x A \alpha^{k+1} \Rightarrow x \gamma \alpha^{k+1} \dots \xRightarrow{*} x z \dots$)

Como $z = z$, y $|z| \geq k$, pero $\gamma \neq B\beta$, entonces usando la definición 7.3, se concluye que la gramática no es $LL(k)$. Esto es absurdo, ya que la hipótesis del teorema era que G es $LL(k)$. Esta contradicción proviene de haber supuesto que G era recursiva a izquierda, por lo que no lo es. \square

7.6. Relación entre Lenguajes LL y Autómatas de Pila Determinísticos

Teorema 7.4. *Sea L un lenguaje $LL(k)$. Entonces, existe un autómata de pila determinístico (APD) M con un solo estado, que reconoce a L , i.e. $L = \mathcal{L}(M)$.*

Demostración. Ver el paper *Notes on top-down languages*, publicado por R. Kurki-Suonio.

Conceptualmente, se demuestra de manera similar al Teorema 5.5, pero usando una gramática LL como la descrita en el Teorema 7.5, y una función de transición $\delta : Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^*$. \square

La idea del autómata de pila del Teorema 7.4 no es *recordar* la cadena que le fue provista inicialmente, sino la descripción actual para lo que falta consumir de la cadena (el símbolo más a la izquierda de esta subcadena está en el tope de la pila). En un paso, el autómata reemplaza el tope por una cadena (posiblemente vacía), y puede aceptar el siguiente símbolo de entrada. El paso a seguir depende del símbolo que se encuentre en el tope de la pila, y de los siguientes símbolos a ser consumidos.

7.7. Resultados sobre lenguajes y gramáticas LL

Los siguientes resultados sobre gramáticas libres de contexto no recursiva serán utilizados para probar el Teorema 7.5:

Proposición 7.1. *Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto no recursiva. Existe una constante c tal que si $A \xRightarrow[L]{i} wB\alpha$, y $|w| = n$, entonces $i \leq c^{n+1}$.*

Demostración. Se demuestra en el Lema 7.2 □

Lema 7.1. *Sean $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto, y $k \geq 1$ un número entero fijo. Se extiende a G con el símbolo no terminal S_k y el símbolo terminal \perp , tales que $S_k \rightarrow S\perp^k \in P$. Entonces, G es $LL(k)$ si, y sólo si, cuando vale que:*

$$\begin{aligned} S_k &\xRightarrow{*} xA\alpha\beta \Rightarrow x\alpha_1\alpha\beta \xRightarrow{*} xz... \\ S_k &\xRightarrow{*} yA\alpha\gamma \Rightarrow y\alpha_2\alpha\gamma \xRightarrow{*} yz... \end{aligned}$$

, con $|z| = k$, y $A\alpha$ deriva una cadena de símbolos terminales de longitud mayor o igual que k , esto implica que $\alpha_1 = \alpha_2$.

Teorema 7.5. *Todo lenguaje L que es $LL(k)$ admite una gramática $LL(k)$ tal que si vale*

$$\begin{aligned} S_k &\xRightarrow{*} xA\alpha \Rightarrow x\alpha_1\alpha \xRightarrow{*} xz... \\ S_k &\xRightarrow{*} yA\beta \Rightarrow y\alpha_2\beta \xRightarrow{*} yz... \end{aligned}$$

con $|z| = k$, entonces $\alpha_1 = \alpha_2$.

Demostración. Suponer que G es una gramática $LL(k)$ para el lenguaje L . En caso de ser necesario, dar por transformada la gramática G considerando nuevos símbolos no terminales para A en caso de que α y β generasen distintos primeros k símbolos terminales, y $S_k \xRightarrow{*} xA\alpha$, $S_k \xRightarrow{*} yA\beta$.

Sea $m(\alpha)$ la longitud de la cadena más corta de símbolos terminales que se puede derivar a partir de α . Sea K el conjunto libre de prefijos de palabras de α tales que $S_k \xRightarrow{*} \dots\alpha\dots$, con $m(\alpha) \geq k$ y tal que $m(\alpha') < k$ para toda α' subcadena propia de α .

Por el Lema 7.2, la cantidad de elementos distintos en K es finita. Así, para cada $\alpha \in K$, considerar los pares $\langle A, \alpha \rangle$ y un nuevo símbolo no terminal A_α . La producción $S_k \rightarrow S\perp^k$ se reemplaza por $S_k \rightarrow S_\perp\perp^k$. Por otro lado, cada producción de la forma $A \rightarrow \beta$ se reemplaza por $A_\alpha \rightarrow \beta_\alpha$, donde β_α se obtiene reemplazando en β sus símbolos no terminales por los correspondientes nuevos símbolos no terminales:

- si $\beta = B$, entonces $\beta_\alpha = B_\alpha$.
- si $\beta = x$, entonces $\beta_\alpha = x$.
- si $\beta = \gamma\delta$, tal que $m(\delta) \geq k$, entonces $\beta_\alpha = \gamma_{s(\delta_\alpha)}\delta_\alpha$, donde $s(\delta_\alpha)$ es el prefijo de δ_α que está en K .

Se puede ver que esta nueva gramática también es $LL(k)$, y que es equivalente a la original: cada secuencia $S_k \xRightarrow{*} \xi_1 \xRightarrow{*} \xi_2 \xRightarrow{*} \dots \xRightarrow{*} \xi_n$ de la nueva gramática, corresponde exactamente a una secuencia $S_k \xRightarrow{*} \alpha_1 \xRightarrow{*} \alpha_2 \xRightarrow{*} \dots \xRightarrow{*} \alpha_n$ en la gramática original, tal que α_i se obtiene de ξ_i borrando los subíndices α de los símbolos no terminales nuevos.

Además, la nueva gramática se construye de manera tal que $S_k \xRightarrow{*} xA_\alpha\xi$ implica que en la gramática original $S_k \xRightarrow{*} xA_\alpha\beta$, con $\alpha\beta$ obtenida borrando los subíndices en ξ .

Suponer ahora (con el objetivo de llegar a una contradicción) que $\xi_1 \neq \xi_2$, y que, con $|z| = k$, en la gramática nueva vale que:

$$\begin{aligned} S_k &\xRightarrow{*} xA_\alpha\xi \Rightarrow x\xi_1\xi \xRightarrow{*} xz... \\ S_k &\xRightarrow{*} yA_\alpha\nu \Rightarrow y\xi_2\nu \xRightarrow{*} yz... \end{aligned}$$

Entonces, en la gramática original vale que $\alpha_1 \neq \alpha_2$, y que con $m(\alpha) \geq k$ y $|z| = k$, ocurre que:

$$\begin{aligned} S_k &\xRightarrow{*} xA_\alpha\beta \Rightarrow x\alpha_1\alpha\beta \xRightarrow{*} xz... \\ S_k &\xRightarrow{*} yA_\alpha\gamma \Rightarrow y\alpha_2\alpha\gamma \xRightarrow{*} yz... \end{aligned}$$

, donde $\alpha_1, \alpha_2, \alpha\beta, \alpha\gamma$ se obtienen de ξ_1, ξ_2, ξ, ν borrando los subíndices.

Esto contradice, por lo visto en el Lema 7.1, el hecho de que una gramática libre de contexto es $LL(k)$ exactamente cuando:

$$\begin{aligned} S_k &\xRightarrow{*} xA_\alpha\beta \Rightarrow x\alpha_1\alpha\beta \xRightarrow{*} xz... \\ S_k &\xRightarrow{*} yA_\alpha\gamma \Rightarrow y\alpha_2\alpha\gamma \xRightarrow{*} yz... \end{aligned}$$

, con $|z| = k$ y $A\alpha$ derivado de una cadena de terminales de longitud mayor o igual que k , implica que $\alpha_1 = \alpha_2$. La contradicción proviene de haber supuesto que $\xi_1 \neq \xi_2$, con lo cual lo contrario es cierto. \square

7.8. Parsing LL(1)

Al momento de hacer el análisis sintáctico de una cadena de un lenguaje LL , se suele extender la gramática $G = \langle V_N, V_T, P, S \rangle$ con un símbolo no terminal inicial S' , y un símbolo terminal $\$,$ con la producción $S' \rightarrow S\$$ para poder reconocer el final de la cadena de entrada.

Un *parser* $LL(1)$ reconoce las cadenas de entrada leyendo **de izquierda a derecha**, haciendo siempre la **derivación más a la izquierda**. El resultado es un árbol de derivación *top-down*: partiendo del símbolo distinguido, hasta llegar a las hojas (los *tokens* de la cadena de entrada).

La idea del *parsing* $LL(1)$ será iterar en la cadena de entrada, hasta que se la reconozca por completo, o se llegue a un error. Para eso:

1. Empezando desde el símbolo inicial, se identifica el *token corriente*, el símbolo a reconocer en cada paso.
2. Se identifica qué producción lo reconoce, utilizando los símbolos directrices.
3. Se toma esa producción, y se reconocen todos los *tokens corrientes* posibles, hasta llegar a un nuevo símbolo no terminal, momento en el que se repite el proceso.

7.8.1. Tabla $LL(1)$

Dada una gramática $G = \langle V_N, V_T, P, S \rangle$, la tabla $LL(1)$ es una matriz M de dimensión $|V_N| \times |V_T \cup \{\$\}|$, en la que

$$M(A, a) = A \rightarrow \beta \Leftrightarrow a \in SD(A \rightarrow \beta)$$

Ejemplo 7.5. $G = \langle \{S, A, B\}, \{a, b, c\}, P, S \rangle$, con P dado por:

$$\begin{aligned} S &\rightarrow AB \\ A &\rightarrow aA \mid ca \\ B &\rightarrow bB \mid cb \end{aligned}$$

Calculando símbolos directrices:

- $SD(A \rightarrow aA) = \{a\}$
- $SD(A \rightarrow ca) = \{c\}$
- $SD(B \rightarrow bB) = \{b\}$
- $SD(B \rightarrow cb) = \{c\}$
- $SD(S \rightarrow AB) = \{a, c\}$

Armando la tabla $LL(1)$:

	a	b	c	$\$$
S	$S \rightarrow AB$		$S \rightarrow AB$	
A	$A \rightarrow aA$		$A \rightarrow ca$	
B		$B \rightarrow bB$	$B \rightarrow cb$	

Algoritmo para construir la tabla $LL(1)$

```

1: for each  $A \rightarrow \alpha \in P$  do
2:   for each  $a \in \text{PRIMEROS}(\alpha)$  do
3:      $M[A, a] \leftarrow M[A, a] \cup \{A \rightarrow \alpha\}$ 
4:   end for
5:   if  $\lambda \in \text{PRIMEROS}(\alpha)$  then
6:     for each  $b \in \text{SIGUIENTES}(A)$  do
7:        $M[A, b] \leftarrow M[A, b] \cup \{A \rightarrow \alpha\}$ 
8:     end for
9:     if  $\$ \in \text{SIGUIENTES}(A)$  then
10:       $M[A, \$] \leftarrow M[A, \$] \cup \{A \rightarrow \alpha\}$ 
11:    end if
12:   end if
13: end for
14: for each  $A, a \mid M[A, a] = \emptyset$  do
15:    $M[A, a] \leftarrow \text{error}$ 
16: end for

```

7.8.2. Algoritmo de *Parsing*

Conceptualmente, para construir un *parser* $LL(1)$ para una gramática $G = \langle V_N, V_T, P, S \rangle$, es:

1. Corregir la gramática si fuera necesario, para eliminar ambigüedades.
2. Construir la tabla $LL(1)$ (i.e. calcular símbolos directrices de todas las producciones).
3. Construir una tabla con tres columnas: **pila**, **cadena** y **producción**.
4. Escribir en la columna **cadena**, la cadena de entrada seguida del símbolo \$.
5. Escribir en la columna **pila** el símbolo inicial S seguido de \$ (tope de la pila).
6. Iterar hasta hacer *match* con \$, consumir toda la cadena y vaciar la pila:
 - Si el tope de la pila es un símbolo terminal y coincide con el *token corriente*, entonces la acción es *match*, y se desapila el tope de la pila.
 - Si el tope de la pila es un símbolo terminal y no coincide con el *token corriente*, entonces se rechaza la cadena.
 - Si el tope de la pila es un símbolo no terminal, elegir la producción que tiene al *token corriente* entre sus símbolos directrices, desapilar el tope de la pila y apilar el cuerpo de dicha producción.
 - En otro caso, se rechaza la cadena.

Más formalmente, el algoritmo consiste en:

- Entrada: matriz M (tabla $LL(1)$), cadena de entrada $w\$$.
- Salida: lista de producciones de la derivación $S \xRightarrow[L]{*}$.

Algoritmo de *parsing* LL(1)

```

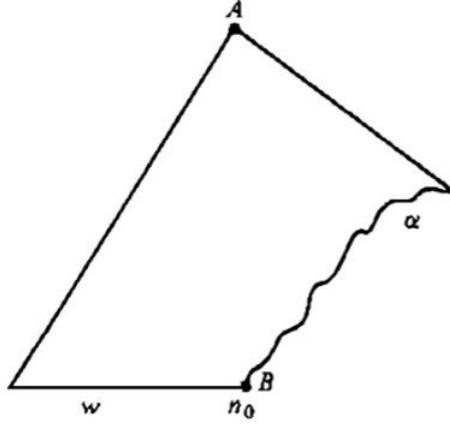
1: apilar  $S\$$ 
2:  $tc \leftarrow$  puntero a la primera posición de  $w\$$ 
3: repeat
4:    $a \leftarrow$  símbolo apuntado por  $tc$ 
5:    $X \leftarrow$  tope de pila
6:   if  $X \in V_T \cup \{\$\}$  then
7:     if  $X = a$  then
8:       desapilar  $X$ 
9:       avanzar  $tc$ 
10:    else
11:      reportar error
12:    end if
13:  else
14:    if  $M[X, a] = (X \rightarrow Y_1Y_2...Y_k)$  then
15:      desapilar  $X$ 
16:      apilar  $Y_1Y_2...Y_k$ 
17:      emitir  $X \rightarrow Y_1Y_2...Y_k$ 
18:    else
19:      reportar error
20:    end if
21:  end if
22: until  $X = \$$ 

```

7.8.3. Complejidad del algoritmo de *parsing* LL(1)

Lema 7.2. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto no recursiva. Existe una constante c tal que si $A \xRightarrow[i]{L} wB\alpha$ y $|w| = n$, entonces $i \leq c^{n+2}$.

Demostración. Sea $k = |V_N|$. Sea \mathcal{A} el árbol de derivación para la derivación más a la izquierda de $A \xRightarrow[i]{L} wB\alpha$.



Sea n_0 el nodo en \mathcal{A} con etiqueta B . Por ser una derivación más a la izquierda, todos los caminos a la derecha del camino desde la raíz hasta n_0 son de longitud menor o igual a este.

Suponer que existe un camino de longitud mayor o igual que $k \times (n + 2)$ nodos desde la raíz a la hoja:

$$A \xRightarrow{L} \alpha_1 \dots \alpha_{k(n+2)} \xRightarrow{L} wB\alpha$$

Considerar la derivación

$$A \xRightarrow{L} \alpha_1 \xRightarrow{L} \dots \xRightarrow{L} \alpha_k$$

$$\alpha_k \xRightarrow{L} \dots \xRightarrow{L} \alpha_{2k}$$

...

$$\alpha_{(n+1)k} \xRightarrow{L} \dots \xRightarrow{L} \alpha_{(n+2)k}$$

Aquí hay $n + 2$ segmentos de k derivaciones. Como $|wB| = n + 1$, es imposible que cada uno de estos segmentos produzca uno o más símbolos de wB . Por lo tanto, hay al menos uno de estos segmentos que no produce ningún símbolo.

Entonces, en el árbol \mathcal{A} hay un camino de nodos n_1, \dots, n_{k+1} que no produce ningún símbolo de wB . Para cada $j = 1, \dots, k$, todos los descendientes directos del nodo n_j que están a la izquierda de n_{j+1} en el árbol, derivan en λ ; es decir que se anulan.

Como cada uno de los nodos n_1, \dots, n_{k+1} tiene como etiqueta a un símbolo no terminal, entonces necesariamente hay dos nodos con la misma etiqueta (recordar que $k = |V_N|$). Sin embargo, esto contradice el hecho de que la gramática no es recursiva a izquierda. La contradicción proviene de suponer que el árbol \mathcal{A} tiene un camino de longitud mayor o igual que $k \times (n + 2)$. Luego, eso no ocurre.

Sea ℓ el máximo número de símbolos en el cuerpo de una producción de G . La cantidad de nodos del árbol de derivación \mathcal{A} es entonces a lo sumo $\ell^{k(n+2)}$. Por lo tanto, si $A \xRightarrow[i]{L} wB\alpha$, entonces $i \leq \ell^{k(n+2)}$. Tomando, $c = \ell^k$, termina la demostración. \square

Corolario. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto no recursiva. Existe una constante c tal que para todo par de símbolos no terminales $A, B \in V_N$, si $A \xRightarrow[i]{L} B\alpha$, entonces $i \leq c^2$.

Observación. El corolario es el caso particular del Lema 7.2 cuando $w = \lambda$.

Teorema 7.6. El algoritmo de parsing $LL(1)$ realiza una cantidad de operaciones lineal en el tamaño de la entrada.

Demostración. La cantidad de pasos que realiza el algoritmo para aceptar una cadena w surge de la cantidad de iteraciones del ciclo. Dado que en cada iteración se desapila un símbolo de la pila, se contará la cantidad de veces que se desapila.

Sea $n = |w|$. El algoritmo requiere que cada uno de los n símbolos terminales sean desapilados.

Dado que la gramática es $LL(1)$, no es recursiva a izquierda. Luego, no hay derivaciones de la forma $A \xRightarrow[L]{+} A\alpha$.

Por lo tanto, basta con acotar la cantidad de pasos en las derivaciones de la forma $A \xRightarrow[L]{+} B\alpha$, con $A \neq B$. Por el Lema 7.2 (en particular, por su corolario), este número de pasos se puede acotar por una constante c . Entonces, el algoritmo desapila a lo sumo $(c + 1) \times n$ veces. \square

7.9. Gramáticas extendidas *ELL*

Las gramáticas extendidas y los métodos *ELL* permiten escribir producciones usando expresiones regulares, y generar *parsers* descendentes (*top-down*) iterativos-recursivos.

Definición 7.6. Una **gramática extendida** es una 4-upla $\langle V_N, V_T, p, S \rangle$, con $p : V_N \rightarrow \text{ER}(V_N \cup V_T)$ (donde $\text{ER}(X)$ es el conjunto de las expresiones regulares que se pueden formar con el conjunto X) una función parcial (puede haber elementos del dominio que no se relacionen con ningún elemento del codominio, ya que no todo símbolo no terminal es cabeza de una producción).

Se puede transformar una gramática *LL* en una extendida *ELL* y viceversa. Además, cada expresión de gramática extendida se corresponde a un fragmento de código para generar el *parser* iterativo-recursivo:

E	$\text{Cod}(E)$	E no extendida
λ	<code>skip;</code>	λ
a	<code>match(a);</code>	a
$R?$	<code>if(tc in Primeros(R)) { Cod(R) }</code>	$X \rightarrow R \mid \lambda$
R^*	<code>while(tc in Primeros(R)) { Cod(R) }</code>	$X \rightarrow RX \mid \lambda$
R^+	<code>do { Cod(R) } while(tc in Primeros(R))</code>	$X \rightarrow RR^*$
$R_1 R_2 \dots R_n$	<code>Cod(R₁); Cod(R₂); ... ; Cod(R_n);</code>	$R_1 R_2 \dots R_n$
$R_1 \mid R_2 \mid \dots \mid R_n$	<code>if(tc in SD(A → R₁)) { Cod(R₁) } elif(tc in SD(A → R₂)) { Cod(R₂) } ... else error</code>	$X \rightarrow R_1 \mid R_2 \mid \dots$

Para transformar una gramática extendida en su gramática no extendida correspondiente (llamada gramática derivada), se sigue el siguiente procedimiento; para cada expresión, reemplazar cada subexpresión que contenga un operador $*$, $|$, $+$, $?$ (yendo desde las más externas a las más internas) por un símbolo no terminal nuevo, agregando las producciones que hagan falta; hacer esto hasta que no queden símbolo por reemplazar.

Observación. Si EG es una gramática extendida y su gramática derivada G es $LL(1)$, se dice que EG es $ELL(1)$. Además, el parser iterativo-recursivo generado con $Cod()$ reconocerá exactamente las cadenas de $\mathcal{L}(EG)$.

7.10. Fuentes

- Verónica Becher. Teoría de Lenguajes, Clase Teórica 9. Primer cuatrimestre, 2022.
- Verónica Becher. Teoría de Lenguajes, Clase Teórica 10. Primer cuatrimestre, 2022.
- Natalia Pesaresi. Teoría de Lenguajes, Clase Práctica 9. Primer cuatrimestre, 2022.
- Sabrina Silvero. Teoría de Lenguajes, Clase Práctica 10. Primer cuatrimestre, 2022.

8. Lenguajes y *Parsing LR*

8.1. Gramáticas $LR(k)$

Las gramáticas $LR(k)$, con $k \geq 0$ un número entero, son gramáticas **libres de contexto no ambiguas**, para las cuales: dada una expresión de $\mathcal{L}(G)$, se puede hallar su **derivación más a la derecha** de forma *bottom-up*, de manera tal que en cada paso de la derivación, la producción elegida queda determinada por los símbolos ya leídos de la cadena de entrada, y k símbolos adicionales más (llamados *lookahead*). De esta forma, cada paso de la derivación se resuelve en tiempo constante.

Los lenguajes $LR(k)$ son exactamente los lenguajes aceptados por **autómatas de pila determinísticos** (APD).

Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto no ambigua. Considerar la gramática extendida $G' = \langle V_N \cup \{S'\}, V_T, P \cup \{S' \rightarrow S\}, S' \rangle$. Sea $w \in \mathcal{L}(G)$; como G es no ambigua, existe una única secuencia de formas sentenciales de G' $\alpha_0, \alpha_1, \dots, \alpha_m$ tal que $\alpha_i \xRightarrow[R]{*} \alpha_{i+1}$, para $i = 0, 1, \dots, m$, con $\alpha_0 = S'$ y $\alpha_m = w$.

El *parsing a derecha* de w es la secuencia de las m producciones usadas en la derivación de w . Las gramáticas $LR(k)$ aseguran que cada α_i es determinable a partir de los k símbolos más de la entrada w además de los ya leídos.

Definición 8.1. (Gramática $LR(k)$) Dada una gramática libre de contexto $G = \langle V_N, V_T, P, S \rangle$, se define la gramática aumentada $G' = \langle V_N \cup \{S'\}, V_T, P \cup \{S' \rightarrow S\}, S' \rangle$. Se dice que G es $LR(k)$, con $k \geq 0$ entero, si las siguientes tres condiciones:

$$\begin{aligned} S' &\xRightarrow[R]{*} \alpha Az \dots \xRightarrow[R]{*} \alpha \beta z \dots \\ S' &\xRightarrow[R]{*} \gamma B \dots \xRightarrow[R]{*} \alpha \beta z \dots \\ |z| &= k \end{aligned}$$

, o bien las siguientes tres condiciones:

$$\begin{aligned} S' &\xRightarrow[R]{*} \alpha Az \xRightarrow[R]{*} \alpha \beta z \\ S' &\xRightarrow[R]{*} \gamma B \dots \xRightarrow[R]{*} \alpha \beta z \\ |z| &< k \end{aligned}$$

, implican que $\alpha = \gamma$ y $A = B$.

Ejemplo 8.1. $G = \langle \{S', S\}, \{a\}, P, S' \rangle$, con P dado por:

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow Sa \mid a \end{aligned}$$

Para decidir si G cumple la definición de $LR(1)$, hay que considerar todo par de derivaciones de uno o más pasos. Hay 3 casos:

1. Sea $\ell > 0$. Considerar el siguiente par de derivaciones:

$$\begin{aligned} S' &\xRightarrow[R]{*} S \xRightarrow[R]{*} a \\ S' &\xRightarrow[R]{*} Sa^\ell \xRightarrow[R]{*} aa^\ell \end{aligned}$$

La única asignación posible para reemplazar en la definición de $LR(1)$ es tomar $\alpha = \gamma = \lambda$, $A = B = S$, $\beta = a$. Como no hay z de longitud 1 coincidente, no hay que verificar esa condición, concluyendo entonces que este par de derivaciones es consistente con la definición de $LR(1)$.

2. Sean $j, \ell > 0$. Considerar el siguiente par de derivaciones:

$$\begin{aligned} S' &\xRightarrow[R]{*} Sa^j \xRightarrow[R]{} aa^j \\ S' &\xRightarrow[R]{*} Sa^{j+\ell} \xRightarrow[R]{} aa^{j+\ell} \end{aligned}$$

Se cumple la condición de $LR(1)$ para $z = a$ tomando $\alpha = \gamma = \lambda$, $A = B = S$, $\beta = a$. Por lo tanto, este par de derivaciones también es consistente con la definición de $LR(1)$.

3. Considerar el siguiente par de derivaciones:

$$\begin{aligned} S' &\xRightarrow[R]{*} S' \xRightarrow[R]{} S \\ S' &\xRightarrow[R]{*} S \xRightarrow[R]{} Sa \end{aligned}$$

No hay un z de longitud 1 coincidente, por lo que no hay que verificar esa condición. Luego, este par de derivaciones también es consistente con la definición de $LR(1)$.

Por lo tanto, se concluye de los tres casos que G es $LR(1)$.

Ejemplo 8.2. $G = \langle \{S', S, A, B\}, \{a, b, c\}, P, S' \rangle$, con P dado por:

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow Ab \mid Bc \\ A &\rightarrow Aa \mid \lambda \\ B &\rightarrow Ba \mid \lambda \end{aligned}$$

Notar que para todo $k \geq 0$, vale que:

$$\begin{aligned} S' &\xRightarrow[R]{*} Aa^k b \xRightarrow[R]{} a^k b \\ S' &\xRightarrow[R]{*} Ba^k c \xRightarrow[R]{} a^k c \end{aligned}$$

Así, se tiene que para $z = a^k$ de longitud k , y $\beta = \lambda$, no se cumple la condición de la definición de $LR(k)$, ya que $A \neq B$.

Notar, por otro lado, que G es recursiva a izquierda, por lo que no puede ser $LL(k)$ para ningún k . Sin embargo, el lenguaje generado por G es $\mathcal{L}(G) = a^*b \mid a^*c$, que es claramente regular, por lo que admite una gramática LL , y también una LR .

8.2. Ambigüedad

Teorema 8.1. *Toda gramática LR es **no ambigua**.*

Demostración. Por el absurdo: suponer que existe una gramática libre de contexto $G = \langle V_N, V_T, P, S \rangle$ tal que G es $LR(k)$ y es ambigua. Luego, por definición existe una cadena w con dos derivaciones (más a la derecha) posibles:

$$\begin{aligned} S &\xRightarrow{R} \alpha_1 \xRightarrow{R} \alpha_2 \xRightarrow{R} \dots \xRightarrow{R} \alpha_n \xRightarrow{R} w \\ S &\xRightarrow{R} \beta_1 \xRightarrow{R} \beta_2 \xRightarrow{R} \dots \xRightarrow{R} \beta_m \xRightarrow{R} w \end{aligned}$$

Sea i el menor índice tal que $\alpha_{n-i} \neq \beta_{m-i}$. Sea $y = \alpha_{n-(i-1)}$ (por la elección de i , también $y = \beta_{m-(i-1)}$). Entonces valen las derivaciones:

$$\begin{aligned} S &\xRightarrow{*R} \alpha_{n-i} \xRightarrow{R} y \\ S &\xRightarrow{*R} \beta_{m-i} \xRightarrow{R} y \end{aligned}$$

Sea $z \in V_T^*$ el sufijo de mayor longitud de α_{n-i} (compuesto únicamente por símbolos terminales, o nulos), que también es sufijo de y . Dado que $\alpha_{n-i} \neq \beta_{m-i}$, por definición es imposible que la gramática G sea $LR(k)$. Esto contradice la hipótesis de que G era $LR(k)$, por lo que era falso que G fuera ambigua. \square

8.3. Relación entre Lenguajes LR y Autómatas de Pila Determinísticos

Teorema 8.2. *Los lenguajes libres de contexto reconocibles por **autómatas de pila determinísticos** (APD) coinciden con los lenguajes $LR(1)$.*

Demostración. No se dió en clase. Ver [A&Uv2], Teoremas 8.10 (p.694) y 8.16 (p.701). \square

8.4. Resultados sobre lenguajes y gramáticas LR

Teorema 8.3. *Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática $LR(k)$, con $k \geq 0$ un número entero. Existe una gramática G' que es $LR(1)$ y tal que $\mathcal{L}(G') = \mathcal{L}(G)$.⁶*

Demostración. No se dió en clase; utiliza varios resultados previos, ver libro para detalles. \square

Teorema 8.4. *Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática $LL(k)$, con $k \geq 0$ un número entero. Si G no tiene producciones inútiles, entonces G es $LR(k)$.⁷*

Demostración. Suponer que la gramática G es $LL(k)$ pero no $LR(k)$. Luego, existen dos derivaciones más a la derecha (en la gramática extendida con S')

$$S' \xRightarrow{iR} \alpha A x_1 \xRightarrow{R} \alpha \beta x_1 \tag{10}$$

$$S' \xRightarrow{jR} \gamma B y \xRightarrow{R} \gamma \delta y \tag{11}$$

, tales que $\gamma \delta y = \alpha \beta x_2$ para algún x_2 para el cual $\text{PRIMEROS}_k(x_2) = \text{PRIMEROS}_k(x_1)$. Como G no es $LR(1)$, se puede suponer que $\alpha A x_2 \neq \gamma B y$.

⁶[A&Uv2] Teorema 8.16 (p.701)

⁷[A&Uv2] Teorema 8.1 (p.669)

Si ocurriese que i y/o j sean 0 se tendría

$$\begin{aligned} S' &\xRightarrow{0} S' \Rightarrow S \\ S' &\xRightarrow{\pm} By \Rightarrow S\beta y \end{aligned}$$

, lo cual implicaría que G es recursiva a izquierda, con lo cual G no podría ser LL . Esto no tendría sentido porque esa era la hipótesis. Por lo tanto, a partir de ahora se puede asumir que tanto i como j son mayores que 0; luego, se puede reemplazar S' por S en las derivaciones (10) y (11).

Sean $x_\alpha, x_\beta, x_\gamma, x_\delta \in V_T^*$ cadenas de símbolos terminales, derivadas a partir de $\alpha, \beta, \gamma, \delta$ respectivamente, y tales que $x_\alpha x_\beta x_2 = x_\gamma x_\delta y$. Considerar las derivaciones más a la izquierda correspondientes a las siguientes derivaciones:

$$S \xRightarrow{*}_R \alpha A x_1 \xRightarrow{*}_R \alpha \beta x_1 \xRightarrow{*}_R x_\alpha x_\beta x_1 \quad (12)$$

y

$$S \xRightarrow{*}_R \gamma B y \xRightarrow{*}_R \gamma \delta y \xRightarrow{*}_R x_\gamma x_\delta y \quad (13)$$

Específicamente sean

$$S \xRightarrow{*}_L x_\alpha A \eta \xRightarrow{*}_L x_\alpha \beta \eta \xRightarrow{*}_L x_\alpha x_\beta \eta \xRightarrow{*}_L x_\alpha x_\beta x_1 \quad (14)$$

y

$$S \xRightarrow{*}_L x_\gamma B \theta \xRightarrow{*}_L x_\gamma \delta \theta \xRightarrow{*}_L x_\gamma x_\delta \theta \xRightarrow{*}_L x_\gamma x_\delta y \quad (15)$$

, donde $\eta, \theta \in (V_N \cup V_T)^*$ son las cadenas adecuadas.

Se puede probar⁸ que la secuencia de pasos en la derivación $S \xRightarrow{*}_L x_\alpha A \eta$ es la secuencia de pasos inicial en la derivación $S \xRightarrow{*}_L x_\gamma B \theta$, o viceversa. Suponer que vale la primera posibilidad (para la segunda, se procede de manera simétrica). Entonces, la derivación (15) se puede escribir como

$$S \xRightarrow{*}_L x_\alpha A \eta \xRightarrow{*}_L x_\alpha \beta \eta \xRightarrow{*}_L x_\gamma B \theta \xRightarrow{*}_L x_\gamma \delta \theta \xRightarrow{*}_L x_\gamma x_\delta \theta \xRightarrow{*}_L x_\gamma x_\delta y \quad (16)$$

Sea \mathcal{T} el árbol de derivación para (16), y sean n_A, n_B los nodos correspondientes a A en $x_\alpha A \eta$ y a B en $x_\gamma B \theta$, respectivamente, como se ve en la Figura 12. Notar que n_B puede ser un descendiente de n_A , y no puede haber superposición entre x_β y x_δ : o son disjuntos, o bien x_δ es subcadena de x_β .

Considerar ahora las dos derivaciones más a la derecha asociadas con \mathcal{T} ; en la primera, \mathcal{T} se expande hacia la derecha hasta el nodo n_A , y en la segunda se expande hasta n_B . Esta última derivación se puede escribir como

$$S \xRightarrow{*}_R \gamma B y \xRightarrow{*}_R \gamma \delta y$$

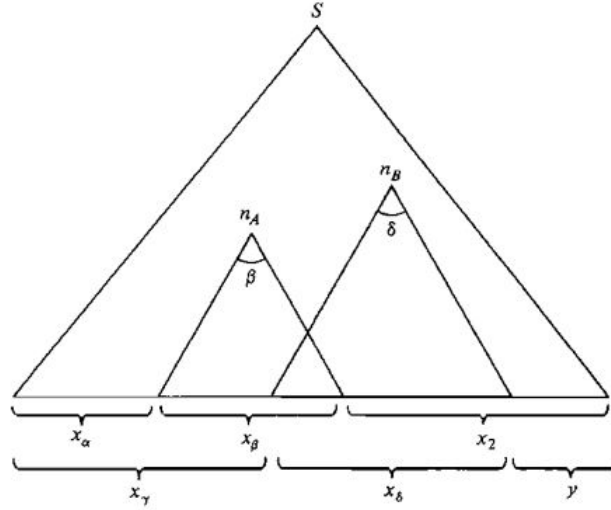
Es decir, la derivación es la misma que (11). La derivación más a la derecha hasta n_A es

$$S \xRightarrow{*}_R \alpha' A x_2 \xRightarrow{*}_R \alpha' \beta x_2 \quad (17)$$

para algún α' . Se probará ahora que $\alpha' = \alpha$.

Suponer que $\alpha' = \alpha$, con el objetivo de llegar a una contradicción sobre el hecho de que G es $LL(k)$. Si $\alpha' = \alpha$, entonces $\gamma \delta y = \alpha' \beta x_2 = \alpha \beta x_2$, con lo cual se puede usar la misma derivación

⁸[A&Uv2] Lema 8.1

Figura 12: Árbol de derivación \mathcal{T} .

más a la derecha para extender las derivaciones (11) y (17) hasta la cadena $x_\gamma x_\delta y \in V_T^*$. Pero como se supuso que los nodos n_A y n_B eran distintos, entonces las derivaciones (11) y (17) son también diferentes, con lo cual sus derivaciones extendidas también lo son. Luego, $x_\gamma x_\delta y$ tiene dos derivaciones más a la derecha diferentes, por lo que G es ambigua. Esto contradice el hecho de que G es $LL(k)$, por lo que la suposición $\alpha' = \alpha$ era falsa⁹.

Ahora se mostrará que $\alpha' = \alpha$. Notar que α' es la cadena formada concatenando las etiquetas de la izquierda de los nodos de \mathcal{T} cuyo ancestro directo es ancestro de n_A . Considerar entonces la derivación más a la izquierda (14), que tiene el mismo árbol de derivación que la derivación más a la derecha (12). Sea \mathcal{T}' el árbol asociado a (12). Los pasos de la derivación (14) hasta llegar a $x_\alpha A \eta$ son los mismos que los de la derivación (16) hasta $x_\alpha A \eta$. Sea n'_A el nodo de \mathcal{T}' correspondiente al símbolo no terminal reemplazado en (16) en el paso $x_\alpha A \eta \xRightarrow{L} x_\alpha \beta \eta$; sea Π un *preorder* de los nodos intermedios del árbol \mathcal{T} (secuencia de los nodos intermedios de \mathcal{T} en el orden en que los nodos se expanden en una derivación más a la izquierda) hasta el nodo n_A , y sea Π' el *preorder* análogo para \mathcal{T}' hasta n'_A . El i -ésimo nodo en Π se *matchea* con el i -ésimo nodo en Π' por el hecho de que ambos nodos tienen la misma etiqueta, y que sus correspondientes descendientes o bien están *matcheados* o están a la derecha de n_A y n'_A , respectivamente.

Los nodos en \mathcal{T}' cuyo ancestro directo es un ancestro de n'_A , tienen etiquetas que, concatenándolas desde la izquierda, forman α . Pero estos nodos están *matcheados* con aquellos en \mathcal{T} que forman α' , con lo cual $\alpha' = \alpha$, completando la demostración. \square

8.5. Parsing LR

El *parsing LR(k)* funciona de manera *bottom-up*, es decir, empezando por las hojas del árbol de derivación (los símbolos de la cadena de entrada), y terminando en la raíz (el símbolo distinguido). Esto es distinto al *parsing top-down* utilizado en lenguajes LL .

La técnica consiste en hacer sucesivas **reducciones**, que lleven desde la cadena de entrada hasta el símbolo distinguido. Como resultado, se obtiene la secuencia (en orden invertido) de derivaciones más a la derecha que producen la cadena dada.

⁹Esto no parece tener sentido, pero al leer la demostración del libro, realmente parece que siempre dice $\alpha' = \alpha$, y nunca $\alpha' \neq \alpha$.

Definición 8.2. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto. Sea la siguiente derivación más a la derecha para la cadena xw :

$$S \xRightarrow{*}_R \alpha Aw \xRightarrow{*}_R \alpha \beta w \xRightarrow{*}_R xw$$

Se dice que la forma sentencial $\alpha \beta w$ puede ser *reducida*, usando la producción $A \rightarrow \beta$, a la forma sentencial αAw .

8.5.1. Pivote

Definición 8.3. Un **pivote** (o *handle*) de una forma sentencial γ es una pareja formada por una producción $A \rightarrow \beta$ y una posición en la cadena γ donde ocurre β .

La **reducción** reemplaza la ocurrencia de β por A , para producir una forma sentencial anterior en la derivación más a la derecha de γ .

Si la derivación es $S' \xRightarrow{*}_R \alpha Aw \xRightarrow{*}_R \alpha \beta w$, entonces el pivote es la producción $A \rightarrow \beta$ y la posición $|\alpha| + 1$.

Observación. La expresión w a la derecha de β es una secuencia de únicamente símbolos terminales (porque las derivaciones son más a la derecha).

La técnica de *parsing* determinista *bottom-up* que opera linealmente consiste en identificar unívocamente el pivote, y hacer una reducción.

Ejemplo 8.3. $G = \langle \{S', S, A, B\}, \{a, b, c, d, e\}, P, S' \rangle$, con P dado por:

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow aABe \\ A &\rightarrow Abc \mid b \\ B &\rightarrow d \end{aligned}$$

Considerar la cadena $w = abbcde$, y su derivación, en la que se subraya el pivote:

$$S' \xRightarrow{*}_R \underline{S} \xRightarrow{*}_R \underline{aABe} \xRightarrow{*}_R aAde \xRightarrow{*}_R a\underline{Abcde} \xRightarrow{*}_R \underline{abbcde}$$

8.5.2. Prefijo viable

Definición 8.4. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto. Se dice que un prefijo de una forma sentencial es **viable** si no continúa más allá del extremo derecho del pivote.

Dada una derivación de la forma:

$$S \xRightarrow{*}_R \alpha Aw \xRightarrow{*}_R \alpha \beta w$$

Todos los prefijos de la cadena $\alpha \beta$ son prefijos viables de la gramática G .

Observación. En el proceso de *parsing bottom-up*, los prefijos viables son las cadenas que se almacenan en la pila.

8.5.3. Ítems $LR(k)$

Definición 8.5. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática $LR(k)$. Un **ítem** de G es una producción con un *punto* '·' en la parte derecha de la producción, y una cadena de símbolos terminales de longitud menor o igual que k .

Un ítem representa *hasta dónde se vio una producción* en el proceso de *parsing*, y cómo se espera que continúe la cadena de entrada.

Ejemplo 8.4. $G = \langle \{S', S, B\}, \{a, b\}, P, S' \rangle$, con P dado por:

$$\begin{aligned} S' &\rightarrow S \\ S &\rightarrow BB \\ B &\rightarrow aB \mid b \end{aligned}$$

Los siguientes son ítems de G :

- $[S' \rightarrow \cdot S, \$]$
- $[B \rightarrow a \cdot B, a]$
- $[B \rightarrow aB \cdot, \$]$

Definición 8.6. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática $LR(k)$. Dada una producción $A \rightarrow \alpha\beta \in P$, un ítem $LR(k)$ $[A \rightarrow \alpha \cdot \beta, u]$, con $u \in V_T^*$ y $|u| \leq k$, es un **ítem válido** para el prefijo viable $\eta\alpha$ si existe una derivación más a la derecha tal que

$$S \xRightarrow[\text{R}]{*} \eta Aw \xRightarrow{\text{R}} \eta\alpha\beta w \quad \text{y } u \in \text{PRIMEROS}_k(w).$$

Con el fin de poder indicarle al *parser* el final de la cadena de entrada, se agregan los ítems con $u = \$$, y se exige que $w = \lambda$.

En el caso $k = 0$, un ítem $LR(0)$ $[A \rightarrow \alpha \cdot \beta]$ es válido para el prefijo viable $\eta\alpha$ si existe una derivación más a la derecha tal que

$$S \xRightarrow[\text{R}]{*} \eta Aw \xRightarrow{\text{R}} \eta\alpha\beta w$$

Teorema 8.5. *El conjunto de prefijos viables de una gramática $LR(1)$ es regular.*

Demostración. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática $LR(1)$. Se define el autómata finito no determinístico con transiciones λ (AFND- λ) $M = \langle Q, V_N \cup V_T, \delta, q_0, Q \rangle$, donde:

- Q es el conjunto de ítems $LR(1)$ válidos de G .
- $q_0 = [S' \rightarrow \cdot S, \$]$
- $\delta : Q \times (V_T \cup V_N \cup \{\lambda\}) \rightarrow \mathcal{P}(Q)$ se define como:
 - $[A \rightarrow \alpha \cdot X\beta, a] \xrightarrow{X} [A \rightarrow \alpha X \cdot \beta, a]$
 - $[A \rightarrow \alpha \cdot B\beta, a] \xrightarrow{\lambda} [B \rightarrow \cdot \gamma, b]$

, para $a \in V_T \cup \{\$, \}$, $A, B \in V_N$, $X \in (V_T \cup V_N)$, $b \in \text{PRIMEROS}(\beta a)$.

Como G es $LR(1)$ (en particular es no ambigua), en cada paso hay una única derivación más a la derecha con un *lookahead* de un símbolo; por lo tanto, existe un autómata finito determinístico $\tilde{M} = \langle \tilde{Q}, V_N \cup V_T, \tilde{\delta}, \tilde{q}_0, \tilde{Q} \rangle$, donde:

- $\tilde{Q} = \{Cl_\lambda(q) : q \in Q \wedge \exists \alpha \in (V_N \cup V_T)^* \mid q \in \hat{\delta}(q_0, \alpha)\}$
- $\tilde{q}_0 = Cl_\lambda([S' \rightarrow .S, \$])$
- $\tilde{\delta} : \tilde{Q} \times (V_N \cup V_T) \rightarrow \tilde{Q}$ definida por:

$$\tilde{\delta}(q, X) = \bigcup_{[A \rightarrow \alpha.X\beta, a] \in q} Cl_\lambda([A \rightarrow \alpha X.\beta, a]), \text{ para } X \in (V_N \cup V_T)$$

En la Figura 13, se puede ver un esquema de la función de transición $\tilde{\delta}$ del AFD \tilde{M} .

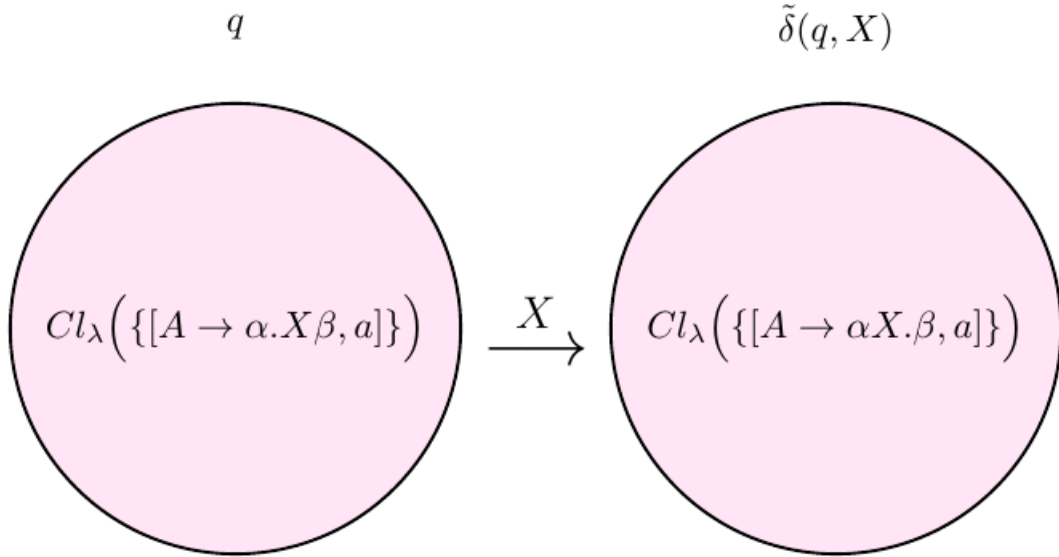
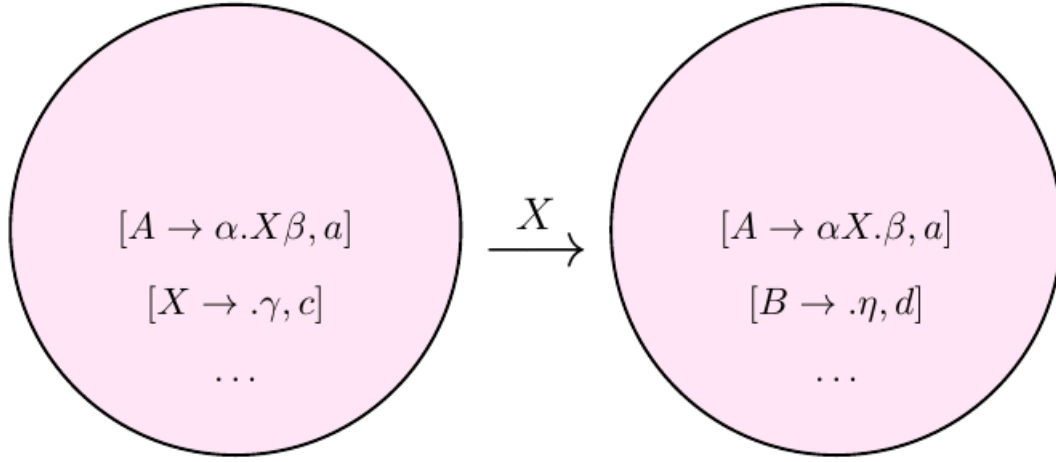


Figura 13: Función de transición $\tilde{\delta}$, con $X \in (V_N \cup V_T)$.

Y en la Figura 14, se puede ver el caso en el que el estado q tiene al menos dos ítems.

$$q = Cl_\lambda\left(\{[A \rightarrow \alpha.X\beta, a]\}\right) \quad \tilde{\delta}(q, X) = Cl_\lambda\left(\{[A \rightarrow \alpha X.\beta, a]\}\right)$$



donde $c \in \text{PRIMEROS}(\beta a)$

si $\beta = B\varphi$ y $d \in \text{PRIMEROS}(\varphi a)$

Figura 14: Función de transición $\tilde{\delta}$, con $X \in V_N$.

Notar que el conjunto de estados finales de \tilde{M} es todo el conjunto de estados \tilde{Q} . Por lo tanto:

$$\mathcal{L}(\tilde{M}) = \{\gamma \in (V_N \cup V_T)^* : \tilde{\delta}(\tilde{q}_0, \gamma) \in \tilde{Q}\}$$

Dado que $q_0 = [S' \rightarrow .S, \$]$, entonces: $[A \rightarrow \alpha.\beta, a] \in \tilde{\delta}(q_0, \gamma)$ si, y sólo si, existe $\eta \in (V_N \cup V_T)^*$ y $w \in V_T^*$ tales que:

- $\gamma = \eta\alpha$
- $S' \xRightarrow{R} S \xRightarrow{R}^* \eta Aw \xRightarrow{R} \eta\alpha\beta w$
- $\text{PRIMEROS}(w) = a$. Notar que es posible que $w = \lambda$ y que $a = \$$.

Luego, el lenguaje aceptado por el AFD \tilde{M} es:

$$\mathcal{L}(\tilde{M}) = \{\gamma \in (V_N \cup V_T)^* : \gamma \text{ es prefijo viable de } G\}$$

, con lo cual es un lenguaje regular. □

Observación. El resultado probado en el Teorema 8.5 se puede generalizar a gramáticas $LR(k)$.

8.6. Parsing LR

Los *parsers LR* se representan como un par de tablas: la *tabla ACCIÓN* y la *tabla IR*. Para construir estas tablas, es necesario armar primero el autómata finito determinístico (AFD) para los prefijos viables.

8.6.1. Algoritmo de Parsing LR(1)

Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática $LR(1)$, y sea $\tilde{M} = \langle \tilde{Q}, (V_N \cup V_T), \tilde{\delta}, \tilde{q}_0, \tilde{Q} \rangle$ un AFD tal que el lenguaje reconocido por \tilde{M} es el conjunto de prefijos viables de G .

Entrada: cadena $a_1a_2\dots a_n\$$

Salida: sucesión de producciones de la derivación más a la derecha de la cadena de Entrada, en orden inverso.

Pila: $q_0X_1q_1\dots X_mq_m$, donde $q \in \tilde{Q}$, $X \in (V_N \cup V_T)$.

Tabla: ACCIÓN: $\tilde{Q} \times (V_T \cup \{\$\}) \rightarrow \{\text{Desplazar } q, \text{Reducir } A \rightarrow \beta, \text{Aceptar}, \text{Error}\}$:

- Si $[A \rightarrow \alpha.a\beta, b] \in q_i$, con $a \in V_T$, e $\text{IR}(q_i, a) = q_j$, entonces $\text{ACCIÓN}(q_i, a) = \text{Desplazar } q_j$.
- Si $[A \rightarrow \alpha., a] \in q_i$, con $a \in V_T \cup \{\$\}$, $A \neq S'$, entonces $\text{ACCIÓN}(q_i, a) = \text{Reducir } A \rightarrow \alpha$.
- Si $[S' \rightarrow S., \$] \in q_i$, entonces $\text{ACCIÓN}(q_i, \$) = \text{Aceptar}$.
- En cualquier otro caso, $\text{ACCIÓN}(q_i, a) = \text{Error}$ (esto significa que se rechaza la cadena de entrada).

Función: $\text{IR}: \tilde{Q} \times (V_N \cup V_T) \rightarrow \tilde{Q}$ es la función de transición $\tilde{\delta}(p, X)$ del AFD \tilde{M} :

- $\text{IR}(q, A)$, para $A \in V_N$, se usa en el algoritmo al apilar un estado en la pila.
- $\text{IR}(q, a)$, para $a \in V_T \cup \{\$\}$, se usa para definir la tabla ACCIÓN.

Algoritmo de *parsing* LR(1)

```

1: apilar  $\tilde{q}_0$ 
2:  $tc \leftarrow$  puntero en la primera posición de la entrada
3: repeat
4:    $q \leftarrow$  estado en el tope de la pila
5:    $a \leftarrow$  símbolo apuntado por  $tc$ 
6:   if  $\text{ACCIÓN}(q, a) = \text{Desplazar } p$  then
7:     apilar  $a$ 
8:     apilar  $p$ 
9:     avanzar  $tc$ 
10:  end if
11:  if  $\text{ACCIÓN}(q, a) = \text{Reducir } A \rightarrow \alpha$  then
12:    desapilar  $2|\alpha|$  símbolos de la pila
13:     $p \leftarrow$  tope de la pila
14:    apilar  $A$ 
15:    apilar  $\text{IR}(p, A)$ 
16:    emitir  $A \rightarrow \alpha$ 
17:  end if
18: until  $\text{ACCIÓN}(q, a) = \text{Aceptar} \vee \text{ACCIÓN}(q, a) = \text{Error}$ 

```

Observación. Este algoritmo es exactamente igual para gramáticas $LR(k)$ en general. Lo que cambia es la construcción de la tabla ACCIÓN. Por ejemplo, en $LR(0)$ no hay *lookahead*, así que la construcción de la tabla es igual, pero los ítems no tienen el símbolo de *lookahead* al final.

8.6.2. Complejidad del algoritmo de *parsing LR(k)*

Para demostrar que el algoritmo de *parsing LR(k)* tiene complejidad lineal, es necesario primero probar ciertos resultados sobre el cómputo en autómatas de pila.

Notar primero que los autómatas finitos (ya sea AFD o AFND) *no se cuelgan*: realizan una cantidad de transiciones exactamente igual a la cantidad de símbolos de entrada. En el caso de los AFND- λ , las transiciones λ no permiten acotar la cantidad de transiciones, pero es posible transformar un AFND- λ en un AFND (que no se cuelga).

Respecto a los autómatas de pila, la cantidad de transiciones realizadas por un APD no está acotada por el tamaño de entrada, dado que depende también del contenido de la pila. Es posible que un APD realice una cantidad infinita de movimientos λ desde alguna configuración. Se dice que esas configuraciones *ciclan*:

Definición 8.7. Sea $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ un autómata de pila determinístico (APD). Una configuración (q, w, α) , con $|\alpha| \geq 1$, cicla si:

$$(q, \mathbf{w}, \alpha) \vdash (p_1, \mathbf{w}, \beta_1) \vdash (p_2, \mathbf{w}, \beta_2) \vdash \dots$$

, con $|\beta_i| \geq |\alpha|$ para todo $i = 1, 2, \dots$

Intuitivamente, una configuración cicla si el APD realiza un número infinito de movimientos sin leer ningún símbolo de la entrada, y el tamaño de la pila es de tamaño mayor o igual de lo que era al principio.

Teorema 8.6. Sea $M = \langle Q, \Sigma, \Gamma, \delta, q_0, Z_0, F \rangle$ un autómata de pila determinístico (APD). Existe un APD M' tal que $\mathcal{L}(M) = \mathcal{L}(M')$, y M' no tiene configuraciones que ciclan.¹⁰

Demostración. Como resultado intermedio, se usa que para un APD cualquiera (en particular, para M):

$$\forall A \in \gamma, \alpha \in \Gamma^* : (q, w, A) \vdash (q', e, e) \Rightarrow (q, w, A\alpha) \vdash (q', e, \alpha)$$

Este resultado (ver [A&Uv1] Lema 2.20 (p.172)) formaliza la noción de que, en un instante dado, lo que sea que se encuentre en el tope de la pila de un APD es independiente de lo que esté *por debajo* de ese tope.

La función de transición de M es $\delta : Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma^*$. Suponer que en cada transición de M , se apilan menos de ℓ símbolos de Γ (la cantidad de símbolos que se apilan debe ser finita en cada paso). Entonces, la cantidad de transiciones sin ciclar es, a lo sumo,

$$\left(|Q| \times |\Gamma|^\ell\right)^{|Q| \times |\Gamma|}$$

A partir de esa cantidad, es posible definir un APD M' que verifica que no pasa dos veces por la misma configuración. \square

¹⁰[A&Uv1] Teorema 2.22 (p.207)

Teorema 8.7. *El algoritmo de parsing LR(k) realiza una cantidad de operaciones lineal en el tamaño de la entrada.*¹¹

Demostración. Sea $a_1 \dots a_n \$$ la cadena de entrada. En cada iteración, el algoritmo realiza la acción **Desplazar** o la acción **Reducir**.

Se define una C -configuración $(q_0 x_1 q_1 \dots x_m q_m, a_i a_{i+1} \dots a_n \$)$ de tres tipos:

- inicial, $(q_0, a_1 \dots a_n \$)$.
- después de una acción **Desplazar**.
- después de una acción **Reducir**, en caso de que la pila haya disminuido su tamaño.

Se le asigna a cada C -configuración el valor

$$v = |\text{símbolos de la pila}| + 3|\text{símbolos por leer}|$$

Así, por ejemplo, la C -configuración inicial tiene valor $v = 1 + 3n$.

Sean C_1 y C_2 dos C -configuraciones consecutivas. Hay dos posibilidades:

- C_2 surge de una acción **Desplazar** (es decir, se leyó un símbolo de la entrada), y su valor es 1 menos que el de C_1 , o
- C_2 surge de una acción **Reducir**, y su valor es 2 menos que el de C_1 , o menor.

Luego, dada una entrada de longitud n , si el algoritmo termina entonces pasa por a lo sumo $1 + 3n$ C -configuraciones.

Ahora se probará que entre dos C -configuraciones hay una cantidad constante de operaciones. Para eso, se simula el algoritmo $LR(k)$ en un autómata de pila determinístico (APD), donde la pila del APD coincide con la del algoritmo.

Por el Teorema 8.6, si a partir de cierto momento un APD no reduce su pila y no lee más de la entrada, entonces está en un ciclo.

Como la gramática es LR , para cada palabra del lenguaje generado por ella existe una única derivación más a la derecha (por ser no ambigua). Tampoco hay recursión a derecha, por lo que el algoritmo de *parsing* no puede pasar dos veces por el mismo par $\langle \text{estado, tope de pila} \rangle$ sin haber leído nuevos símbolos de la entrada. Por lo tanto, se puede afirmar que el algoritmo no cicla.

Dado que la función de transición del APD es $\delta : Q \times \Sigma \times \Gamma \rightarrow Q \times \Gamma$, entonces suponiendo que en cada transición del APD se apilan menos de ℓ símbolos de Γ , se tiene que la máxima cantidad de transiciones del APD sin ciclar, y sin leer de la cadena de entrada, es a lo sumo

$$(|Q| \times |\Gamma|^\ell)^{|Q| \times |\Gamma|}$$

Se concluye entonces que el APD que simula el algoritmo de *parsing* $LR(k)$ no entra en un ciclo, por lo que la cantidad total de pasos en la derivación para aceptar la cadena de entrada es lineal en la longitud de dicha cadena. \square

¹¹[A&Uv1] Teorema 5.13

8.6.3. Conflictos LR y variantes del algoritmo

Al momento de intentar realizar un *parsing* de una cadena mediante el algoritmo de *parsing LR(k)*, pueden surgir **conflictos** de dos tipos:

1. Shift-reduce: en un paso dado, se puede realizar la acción **Desplazar** o la acción **Reducir**.
2. Reduce-reduce: en un paso dado, hay dos producciones de la gramática por las cuales se puede realizar la acción **Reducir**.

Existen variantes de los algoritmos de *parsing LR*, que toman decisiones levemente distintas para evitar estos conflictos. Por ejemplo, el *parsing SLR* (*Simple LR*) es una variante de $LR(0)$, que en vez de reducir en todas las columnas de la tabla ACCIÓN como esta última, reduce sólo para aquellos símbolos terminales que pertenezcan a SIGUIENTES del símbolo no terminal de la cabeza de la producción a reducir. Para las otras columnas, se rechaza la cadena. Intuitivamente, esto es para *no hacer reducciones que no tendrían sentido*.

Por otro lado, en el algoritmo de *parsing LR(1)*, el autómata finito de los prefijos viables suele tener una cantidad de estados grande, con lo cual las tablas también son grandes. Para achicar un poco estas dimensiones, existe el *parsing LALR*, cuyo autómata tiene la misma cantidad de estados que los de *SLR* o $LR(0)$, pero mantiene información contextual como $LR(1)$. La idea del autómata *LALR* es unir los estados cuyos ítems tienen las mismas producciones pero distintos símbolos de *lookahead*. Sin embargo, esta disminución en la cantidad de estados trae como costo que pueden surgir conflictos reduce-reduce que no aparecerían en la alternativa $LR(1)$.

Por último, vale la siguiente relación de inclusiones:

$$LR(0) \subset SLR \subset LALR \subset LR(1) \subset \text{GLC no ambiguas} \subset \text{GLC}$$

8.7. Fuentes

- Verónica Becher. Teoría de Lenguajes, Clase Teórica 11. Primer cuatrimestre, 2022.
- Elisa Orduna. Teoría de Lenguajes, Clase Práctica 12. Primer cuatrimestre, 2022.
- Elisa Orduna. Teoría de Lenguajes, Clase Práctica 13. Primer cuatrimestre, 2022.
- Manuel Panichelli. Teoría de Lenguajes, Clase Práctica 14. Primer cuatrimestre, 2022.

9. Parsing generalizado para Lenguajes Libres de Contexto

Para lenguajes libres de contexto en general, no existen algoritmos de *parsing* tan eficientes como los estudiados para lenguajes *LL* o *LR* (que tenían complejidad lineal). Para estos casos, se utilizan algoritmos de *parsing* que funcionan en casos generales de lenguajes libres de contexto, como los que se verán a continuación.

9.1. Parsing de Cocke-Younger-Kasami

El algoritmo de Cocke-Younger-Kasami (**CYK**) realiza una cantidad de operaciones en el orden de n^3 (consumiendo espacio en memoria del orden de n^2), siendo n el tamaño de la entrada. Se podría decir que se basa en programación dinámica; debido a su eficiencia, no es de gran utilidad práctica (hay algoritmos más eficientes), pero se lo estudia por su importancia teórica e histórica.

Para empezar a describir el algoritmo, primero se deberá recordar el concepto de *forma normal de Chomsky*:

Definición 9.1. Una gramática libre de contexto (GLC) $G = \langle V_N, V_T, P, S \rangle$ está en **forma normal de Chomsky** si todas sus producciones son de la forma $A \rightarrow BC$ o $A \rightarrow a$, para $A, B, C \in V_N$, $a \in V_T$; y si $\lambda \in \mathcal{L}(G)$, entonces $S \rightarrow \lambda \in P$, pero S no aparece en el cuerpo de ninguna producción de G .

Dicho de otra manera, las GLCs en forma normal de Chomsky son aquellas cuyas producciones tienen a lo sumo dos símbolos en su cuerpo. Esto las hace particularmente convenientes para trabajar, y vale que toda gramática libre de contexto se puede transformar a forma normal de Chomsky, usando el siguiente algoritmo.

9.1.1. Algoritmo para pasar a forma normal de Chomsky

Antes de mostrar el algoritmo en cuestión, se describirá un algoritmo para *remover producciones unitarias*:

Algoritmo para remover producciones unitarias¹²

- Entrada: gramática libre de contexto $G = \langle V_N, V_T, P, S \rangle$, sin producciones λ .
- Salida: gramática libre de contexto $G = \langle V_N, V_T, P', S \rangle$, sin producciones λ y sin producciones unitarias, tal que $\mathcal{L}(G) = \mathcal{L}(G')$.
- Procedimiento:
 1. Para cada $A \in V_N$, construir el conjunto $N_A = \{B \in V_N : A \xRightarrow{*} B\}$ de la siguiente manera:
 - a) Sean $N_{-1} := \emptyset$, $N_0 := \{A\}$, e $i := 0$.
 - b) Mientras $N_i \neq N_{i-1}$, incrementar i y definir N_i como $N_{i-1} \cup \{C \in V_N : B \rightarrow C \in P \wedge B \in N_{i-1}\}$.

¹²[A&Uv1] Algoritmo 2.11 (p.149)

2. Sea $N_A := N_i$
3. Construir el conjunto de producciones P' de la siguiente manera:
 - Si $B \rightarrow \alpha \in P$, y no es una producción unitaria, entonces agregar $A \rightarrow \alpha$ a P' , para todo A tal que $B \in N_A$.

Ahora sí, la gramática G' sin producciones unitarias será entrada para el algoritmo usado para pasar la gramática a forma normal de Chomsky:

Algoritmo para pasar a forma normal de Chomsky¹³

- Entrada: gramática libre de contexto $G = \langle V_N, V_T, P, S \rangle$, sin producciones unitarias.
- Salida: gramática libre de contexto $G' = \langle V'_N, V_T, P', S' \rangle$ en forma normal de Chomsky, tal que $\mathcal{L}(G) = \mathcal{L}(G')$.
- Procedimiento: definir el conjunto de producciones P' de la siguiente manera:

1. Agregar $S' \rightarrow S$.
2. Si $\lambda \in \mathcal{L}(G)$, entonces agregar $S' \rightarrow \lambda$.
3. Para cada producción $A \rightarrow X_1 X_2 \in P$, con X_1, X_2 , o ambos en V_T (i.e. ya está en forma correcta), agregar

$$A \rightarrow X'_1 X'_2$$

4. Para cada producción $A \rightarrow X_1 \dots X_k \in P$, con $k > 2$ y $X_i \in V_N \cup V_T$ (i.e. no está en forma correcta), agregar

$$\begin{aligned} A &\rightarrow X'_1 \langle X_2 \dots X_k \rangle \\ \langle X_2 \dots X_k \rangle &\rightarrow X'_2 \langle X_3 \dots X_k \rangle \end{aligned}$$

...

$$\begin{aligned} \langle X_{k-2} \dots X_k \rangle &\rightarrow X'_{k-2} \langle X_{k-1} X_k \rangle \\ \langle X_{k-1} X_k \rangle &\rightarrow X'_{k-1} X'_k \end{aligned}$$

5. Si $X_i \in V_T$, entonces agregar $X'_i \rightarrow X_i$, pues X'_i es nuevo (hay que agregarlo también a V'_N).
 6. Si $X_i \in V_N$, entonces $X'_i := X_i$.
 7. Para $j = 1, 2, \dots, k-1$, son nuevos los símbolos no terminales $\langle X_j \dots X_k \rangle$ (tuplas con los símbolos originales), así que hay que agregarlos a V'_N .
- Complejidad: observar que $|V'_N| \leq |V_N| + |V_T| + \ell \times |P|$, donde ℓ es la máxima cantidad de símbolos en el cuerpo de las producciones de P . El algoritmo realiza una cantidad de operaciones lineal en $|P|$ (hace ℓ operaciones básicas por cada producción, siendo ℓ una constante de la gramática).

9.1.2. Algoritmo de construcción de la tabla CYK

Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto en forma normal de Chomsky, y sin producciones λ . Sea $w = a_1 a_2 \dots a_n$ la cadena de entrada. Observar que por la precondition, la gramática G no tiene ciclos.

¹³[A&Uv1] Algoritmo 2.12 (p.151)

El algoritmo CYK construye una tabla triangular \mathcal{T} para w :

$$\mathcal{T} = (t_{i,j})_{1 \leq i \leq n, 1 \leq j \leq n-i+1}$$

Cada $t_{i,j}$ es un subconjunto del conjunto de símbolos no terminales V_N , definido por:

$$A \in t_{i,j} \Leftrightarrow A \xRightarrow{+} a_i a_{i+1} \dots a_{i+j-1}$$

La definición de cada $t_{i,j}$ es:

$$t_{i,j} = \{A \in V_N : A \rightarrow BC \in P, \text{ para algún } k, 1 \leq k < j, B \in t_{i,k}, C \in t_{i+k,j-k}\}$$

Para el caso $j = 1$, se tiene:

$$t_{i,1} = \{A \in V_N : A \rightarrow a_i \in P\}$$

Por lo tanto,

$$w \in \mathcal{L}(G) \Leftrightarrow S \in t_{1,n}$$

En caso de querer las derivaciones de w , se pueden reconstruir usando la tabla \mathcal{T} (notar que se construye una tabla por cada cadena de entrada).

Algoritmo Table Cocke-Younger-Kasami¹⁴

- Entrada: gramática libre de contexto $G = \langle V_N, V_T, P, S \rangle$ en forma normal de Chomsky, sin producciones λ , y $w = a_1 a_2 \dots a_n \in V_T^+$ la cadena de entrada a *parsear*.
- Salida: tabla \mathcal{T} para w , tal que $t_{i,j}$ contiene a A si, y sólo si, $A \xRightarrow{+} a_i a_{i+1} \dots a_{i+j-1}$.
- Procedimiento:

1. Definir $t_{i,1} := \{A \in V_N : A \rightarrow a_i \in P\}$, para $i = 1, \dots, n$.
2. Si ya se calculó $t_{i,j'}$ para $i = 1, \dots, n$ y $j' = 1, \dots, j-1$, entonces definir

$$t_{i,j} = \{A \in V_N : A \rightarrow BC \in P, \text{ para algún } k, 1 \leq k < j, B \in t_{i,k}, C \in t_{i+k,j-k}\}$$

Notar que $t_{i,k}$ y $t_{i+k,j-k}$ se calculan antes que $t_{i,j}$ porque $1 \leq k < j$ (el último *menor* es estricto), por lo que $k < j$ y $j-k < j$.

Notar también que si $A \in t_{i,j}$ entonces:

$$A \Rightarrow BC \xRightarrow{+} a_i \dots a_{i+k-1} C \xRightarrow{+} a_i \dots a_{i+k-1} a_{i+k} \dots a_{i+j-1}$$

3. Repetir hasta completar $j = n - i + 1$.

Ejemplo 9.1. $G = \langle \{S, A\}, \{a, b\}, P, S \rangle$, con P definido por:

$$S \rightarrow AA$$

$$S \rightarrow AS$$

$$S \rightarrow b$$

$$A \rightarrow SA$$

$$A \rightarrow AS$$

$$A \rightarrow a$$

La tabla \mathcal{T} para la cadena $w = abaab$ es:

¹⁴[A&Uv1] Algoritmo 4.3

j=5	A,S				
j=4	A,S	A,S			
j=3	A,S	S	A,S		
j=2	A,S	A	S	A,S	
j=1	A	S	A	A	S
	i= 1	i=2	i=3	i=4	i=5

Recordar que $\mathcal{T} = (t_{i,j})_{1 \leq i \leq n, 1 \leq j \leq n-i+1}$, con

$$t_{i,j} = \{X \in \{S, A\} : X \rightarrow BC \in P, B \in t_{i,k}, C \in t_{i+k,j-k}, 1 \leq k < j\}$$

Dado que $a_1 = a_3 = a_4 = a$, y $a_2 = a_5 = b$, se tiene:

$$t_{1,1} = t_{3,1} = t_{4,1} = \{A\}$$

$$t_{2,1} = t_{5,1} = \{S\}$$

$$t_{1,2} = \{S, A : S \rightarrow AS, A \rightarrow AS, A \in t_{1,1}, S \in t_{2,1}\}$$

$$t_{2,2} = \{A : A \rightarrow SA, S \in t_{2,1}, A \in t_{3,1}\}$$

$$t_{3,2} = \{S : S \rightarrow AA, A \in t_{3,1}, A \in t_{4,1}\}$$

...

$$t_{1,3} = \{S : S \rightarrow AA, A \in t_{1,1}, A \in t_{2,2}\} \cup \{A, S : S \rightarrow AA, A \rightarrow SA, A, S \in t_{1,2}, A \in t_{3,1}\}$$

$$t_{2,3} = \{S : S \rightarrow AA, A \in t_{2,2}, A \in t_{4,1}\}$$

Notar que $S \in t_{2,1}$, $S \in t_{3,2}$, pero no hay una producción de la forma $X \rightarrow SS$ para agregar a $t_{2,2}$.

9.1.3. Corrección de la tabla

Teorema 9.1. ¹⁵ Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto en forma normal de Chomsky, y sin producciones λ . Sea \mathcal{T} la tabla del Algoritmo CYK para la cadena de entrada $a_1 \dots a_n$. Entonces

$$A \in t_{i,j} \Leftrightarrow A \Rightarrow^+ a_i \dots a_{i+j-1}$$

Demostración. Suponer que vale el resultado para todo i ; procediendo por inducción en $j > 0$:

- Caso base: $j = 1$. Por definición de \mathcal{T} en el caso $j = 1$, vale que $A \in t_{i,1} \Leftrightarrow A \rightarrow a_i$.
- Paso inductivo: $j \geq 2$. Suponer que vale el resultado para $j - 1$. Por definición de \mathcal{T} , se tiene que:

$$t_{i,j} = \{A \in V_N : A \rightarrow BC \in P, \text{ para algún } k, 1 \leq k < j, B \in t_{i,k}, C \in t_{i+k,j-k}\} \quad (18)$$

Viendo las dos implicaciones por separado:

- \Rightarrow) Suponer $A \in t_{i,j}$. Luego, por (18) se tiene que $A \Rightarrow BC$ para algún $k, 1 \leq k < j$, $B \in t_{i,k}$ y $C \in t_{i+k,j-k}$.

¹⁵[A&Uv1] Teorema 4.6

Por hipótesis inductiva, vale que:

$$\begin{aligned} B &\stackrel{\pm}{\Rightarrow} a_i \dots a_{i+k-1}, \quad y \\ C &\stackrel{\pm}{\Rightarrow} a_{i+k} \dots a_{i+k+j-k-1} \end{aligned}$$

Entonces:

$$A \stackrel{\pm}{\Rightarrow} a_i \dots a_{i+j-1}$$

- \Leftarrow) Suponer ahora que $A \stackrel{\pm}{\Rightarrow} a_i \dots a_{i+j-1}$. Entonces:

$$A \Rightarrow BC \stackrel{\pm}{\Rightarrow} a_i \dots a_{i+j-1}$$

, con $B \stackrel{\pm}{\Rightarrow} a_i \dots a_{i+k-1}$ y $C \stackrel{\pm}{\Rightarrow} a_{i+k} \dots a_{i+j-1}$.

Por hipótesis inductiva, $B \in t_{i,k}$ y $C \in t_{i+k,j-k}$. Por lo tanto, $A \in t_{i,j}$.

Con eso queda probada la doble implicación, y así también el paso inductivo.

Con lo cual, vale el resultado para todo $j > 0$. □

9.1.4. Complejidad de la construcción de la tabla

Teorema 9.2. ¹⁶ *La construcción de la tabla \mathcal{T} del Algoritmo CYK, para una cadena de largo n , realiza una cantidad de operaciones en el orden de n^3 .*

Demostración. Sea $\mathcal{T} = (t_{i,j})_{1 \leq i \leq n, 1 \leq j \leq n-i+1}$. Para fijar el valor de $t_{1,1}, t_{2,1}, \dots, t_{n,1}$, hacen falta n inspecciones en P , el conjunto de producciones de la gramática del algoritmo.

Sea j fijo; en el paso j se debe determinar el valor de $t_{i,j}$ para $i = 1, \dots, n - j + 1$. En total son $n - j + 1$ conjuntos.

Sea i fijo. Por definición de la tabla, para determinar $t_{i,j}$, es necesario examinar $t_{i,k}$ y $t_{i+k,j-k}$, para $k = 1, \dots, j - 1$, que ya fueron computados antes. En total son $2(j - 1)$ conjuntos.

Recordar que:

$$\sum_{j=1}^n j = \frac{n(n+1)}{2} \quad y \quad \sum_{j=1}^n j^2 = \frac{n(n+1)(2n+1)}{6}$$

Luego,

$$\begin{aligned} \sum_{j=1}^n (n - j + 1) 2(j - 1) &= 2 \sum_{j=1}^n (n + 2)j - n - j^2 - 1 \\ &= \frac{n(n^2 - 1)}{3} \in \mathcal{O}(n^3) \end{aligned}$$

□

9.1.5. Algoritmo de *parsing* CYK

Visto cómo construir la tabla \mathcal{T} , se verá ahora el algoritmo CYK para *parsear* una cadena para un lenguaje libre de contexto.

¹⁶[A&Uv1] Teorema 4.7

Algoritmo Cocke-Younger-Kasami¹⁷

- Entrada: gramática libre de contexto $G = \langle V_N, V_T, P, S \rangle$ en forma normal de Chomsky, con las producciones numeradas con números desde 1 a p ; $w = a_1 \dots a_n$ cadena de entrada; tabla \mathcal{T} para w .
- Salida: derivación más a la izquierda para w , o **Error**.
- Procedimiento: Se define un procedimiento recursivo $gen(i, j, A)$ para generar una derivación correspondiente a $A \xRightarrow{*}_L a_i \dots a_{i+j-i}$. Asumiendo que $S \in t_{1,n}$, el algoritmo debe ejecutar $gen(1, n, S)$; es decir, $gen(i, j, A)$ con $i = 1, j = n, A = S$.
 - Caso base: si $j = 1$, $A \in t_{i,1}$, $A \rightarrow a_i \in P$, emitir el número de esa producción.
 - Caso recursivo: si $j > 1$ y $A \in t_{i,j}$, sea k el mínimo índice tal que $1 \leq k < j$ y hay $B \in t_{i,k}$ y $C \in t_{i+k,j-k}$, $A \rightarrow BC \in P$ (podría haber varias). Emitir el número de esta producción y ejecutar $gen(i, k, B)$ seguido de $gen(i+k, j-k, C)$.

9.1.6. Complejidad del Algoritmo CYK

Teorema 9.3. ¹⁸ *El Algoritmo de parsing CYK para una cadena de entrada de longitud n , realiza una cantidad de operaciones en el orden de n^2 .*

Demostración. Sea $G = \langle V_N, V_T, P, S \rangle$ en forma normal de Chomsky, y \mathcal{T} la tabla CYK para la cadena w de longitud n .

Se probará por inducción en j que $gen(i, j, A)$ requiere a lo sumo $c_1 \times j^2$ operaciones, siendo c_1 una constante:

- Caso base: $j = 1$. *Trivial*.
- Paso inductivo: $j \geq 2$. Para los valores de k entre 1 y $j - 1$, revisar $t_{i,k}$ y $t_{i+k,j-k}$. Esto lleva $c_2 \times j$ operaciones, para una constante c_2 .

Por hipótesis inductiva, $gen(i, k, B)$ requiere $c_1 \times k^2$ operaciones, y $gen(i+k, j-k, B')$ requiere $c_1 \times (j-k)^2$. Entonces

$$c_1 \times k^2 + c_1 \times (j-k)^2 + c_2 \times j = c_1 \times (j^2 + 2k^2 - 2jk) + c_2 \times j$$

Como $1 \leq k < j$ y $j \geq 2$, se tiene que

$$2k^2 - 2jk \leq 2 - 2j \leq -j$$

Entonces, tomando $c_1 = c_2$ en la hipótesis inductiva, se obtiene que la cantidad de operaciones en el paso j es

$$c_1 \times (j^2 + 2k^2 - 2jk) + c_2 \times j \leq c_1 \times j^2$$

□

¹⁷[A&Uv1] Algoritmo 4.4

¹⁸A&Uv1] Teorema 4.8

9.2. Parsing de Earley

Dada una gramática libre de contexto, el algoritmo de *parsing* de Earley realiza una cantidad de operaciones en el orden de n^3 , y usa espacio en memoria del orden de n^2 , siendo n la longitud de la cadena de entrada.

Sin embargo, una ventaja es que **si la gramática es no ambigua**, la cantidad de operaciones es del orden de n^2 . Además, para la mayoría de gramáticas que definen lenguajes de programación, es posible modificar el algoritmo para obtener una cantidad de operaciones y espacio lineal en la longitud de entrada.

Es un método más similar a lo visto en *parsing LR*, dado que funciona de manera *bottom-up*, y encuentra una derivación más a la derecha.

9.2.1. Ítem de Earley

Definición 9.2. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto, y sea $w = a_1 a_2 \dots a_n \in V_T^*$ una cadena de entrada. Se dice que $[A \rightarrow X_1 X_2 \dots X_k \cdot X_{k+1} \dots X_m, i]$ es un **ítem** para w si $A \rightarrow X_1 X_2 \dots X_m \in P$ es una producción de la gramática, y $0 \leq i \leq n$.

Observación. Si la producción es $A \rightarrow \lambda$, entonces el ítem es $[A \rightarrow \cdot, i]$.

9.2.2. Algoritmo de *parsing* de Earley

Dada $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto, y una cadena de entrada $w = a_1 \dots a_n \in V_T^*$, se construyen las listas de ítems $\ell_1, \ell_2, \dots, \ell_n$ tal que

$$[A \rightarrow \alpha \cdot \beta, i] \in \ell_j \Leftrightarrow \exists \gamma, \delta \mid S \xRightarrow{*}_R \gamma A \delta, \quad \gamma \xRightarrow{*}_R a_1 \dots a_i \quad \text{y} \quad \alpha \xRightarrow{*}_R a_{i+1} \dots a_j$$

Luego, $w \in \mathcal{L}(G)$ si, y sólo si, hay un α tal que $[S \rightarrow \alpha \cdot, 0] \in \ell_n$.

Algoritmo de Earley¹⁹

- Entrada: $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto, cadena de entrada $w = a_1 \dots a_n \in V_T^*$.
- Salida: listas $\ell_1, \ell_2, \dots, \ell_n$.
- Procedimiento:
 1. Si $S \rightarrow \alpha$ es una producción en P , entonces agregar $[S \rightarrow \cdot \alpha, 0]$ a ℓ_0 .
 2. Si $[A \rightarrow \alpha \cdot B \beta, 0] \in \ell_0$ y $[B \rightarrow \gamma \cdot, 0] \in \ell_0$ (en particular, γ puede ser λ), entonces agregar $[A \rightarrow \alpha B \cdot \beta, 0]$ a ℓ_0 .
 3. Si $[A \rightarrow \alpha \cdot B \beta, 0] \in \ell_0$, entonces agregar $[B \rightarrow \cdot \gamma, 0]$ a ℓ_0 , para toda producción $B \rightarrow \gamma \in P$, si es que aún no se agregó.
 4. Suponiendo construidas $\ell_1, \ell_2, \dots, \ell_{j-1}$, entonces: si $[B \rightarrow \alpha \cdot a B \beta, i] \in \ell_{j-1}$ tal que $a = a_j$, entonces agregar $[B \rightarrow \alpha a \cdot \beta, i]$ a ℓ_j .
 5. Si $[B \rightarrow \alpha \cdot A \beta, k] \in \ell_i$ y $[A \rightarrow \gamma \cdot, i] \in \ell_j$, entonces agregar $[B \rightarrow \alpha A \cdot \beta, k]$ a ℓ_j .

¹⁹[A&Uv1] Algoritmo 4.5

6. Si $[A \rightarrow \alpha \cdot B\beta, i] \in \ell_j$, entonces agregar $[B \rightarrow \cdot \gamma, j]$ a ℓ_j , para toda producción $B \rightarrow \gamma \in P$.

Observación. La consideración de un ítem con un símbolo terminal a la derecha del punto '.' no produce nuevos ítems en los pasos 2, 3, 5 y 6.

9.2.3. Corrección del algoritmo

Teorema 9.4. ²⁰ Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto. Entonces $[S \rightarrow \alpha \cdot, 0] \in \ell_n$ si, y sólo si, $S \rightarrow \alpha \in P$ y $\alpha \xrightarrow[R]{*} a_1 \dots a_n$.

Demostración. No se dió en clase, ver libro. □

9.2.4. Derivación a derecha de Earley

Este es un algoritmo para obtener una derivación más a la derecha para la cadena de entrada *parseada*.

Derivación a derecha del Algoritmo de Earley²¹

- Entrada: gramática libre de contexto $G = \langle V_N, V_T, P, S \rangle$ sin ciclos, con las producciones numeradas con números entre 1 y p ; cadena de entrada $w = a_1 \dots a_n$, y listas $\ell_1, \ell_2, \dots, \ell_n$.
- Salida: derivación más a la derecha de w , o **Error**.
- Procedimiento: si $[S \rightarrow \alpha \cdot, 0] \in \ell_n$, entonces $w \in \mathcal{L}(G)$. Fijando una variable global $\pi := \lambda$, ejecutar la rutina $R([S \rightarrow \alpha \cdot, 0], n)$, que se define a continuación, y emitir la secuencia de producciones π .

Rutina $R([A \rightarrow X_1 \dots X_m \cdot, i], j)$

```

1:  $\pi \leftarrow h \times \pi$ , con  $h$  el número de la producción  $A \rightarrow X_1 \dots X_m$ 
2:  $k \leftarrow m$ 
3:  $j' \leftarrow j$ 
4: while  $k > 0$  do
5:   if  $X_k \in V_T$  then
6:      $k \leftarrow k - 1$ 
7:      $j' \leftarrow j' - 1$ 
8:   else ▷ i.e.  $X_k \in V_N$ 
9:      $item \leftarrow [X_k \rightarrow \gamma \cdot, i'] \in \ell_{j'}$ , con  $i'$  algún índice tal
       que  $[A \rightarrow X_1 X_2 \dots X_{k-1} \cdot X_k \dots X_m, i] \in \ell_{i'}$ 
10:    Ejecutar  $R(item, j')$ 
11:     $k \leftarrow k - 1$ 
12:     $j' \leftarrow i'$ 
13:   end if
14: end while
    
```

²⁰[A&Uv1] Teorema 4.9

²¹[A&Uv1] Algoritmo 4.6

9.2.5. Complejidad del algoritmo

Teorema 9.5. ²² Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática libre de contexto, y sea $w \in V_T^*$ una entrada de longitud n . Si G es no ambigua, entonces el Algoritmo de Earley construye las listas $\ell_1, \ell_2, \dots, \ell_n$ en una cantidad de operaciones del orden de n^2 . Si G es ambigua, la cantidad es del orden de n^3 .

Demostración. (Idea) Cada lista ℓ_j tiene a lo sumo $(j+1)k$ ítems, siendo k una constante. Se mantiene una lista enlazada entre todos estos ítems, y una tabla indicando si una producción ya está o no.

Además, se mantiene otra lista enlazada entre los ítems que comparten el símbolo a la derecha del punto '·'; esto sirve para aquellos pasos en los que se requiere buscar los ítems que tienen un determinado símbolo a la derecha del punto.

Cuando la gramática es no ambigua, cada ítem se considera una única vez, y hay una cantidad c constante de operaciones a realizar para cada ítem. Los pasos 4, 5 y 6 del algoritmo aseguran que la cantidad de operaciones es a lo sumo

$$\begin{aligned} c \sum_{j=0}^n (j+1)k &= \frac{ck(n+1)(n+2)}{2} \\ &= c'n^2 \in \mathcal{O}(n^2) \end{aligned}$$

□

Teorema 9.6. ²³ Si existe una derivación más a la derecha para $w = a_1 \dots a_n$, entonces el algoritmo de Earley la encuentra en una cantidad de operaciones en el orden de n^2 .

Demostración. Recordar que, por definición de las listas ℓ_1, \dots, ℓ_n :

$$[A \rightarrow \alpha \cdot \beta, i] \in \ell_j \Leftrightarrow \exists \gamma, \delta \mid S \xRightarrow{*}_R \gamma A \delta, \quad \gamma \xRightarrow{*}_R a_1 \dots a_i \quad \text{y} \quad \alpha \xRightarrow{*}_R a_{i+1} \dots a_j$$

En ℓ_j , unir todos los ítems que tengan la misma segunda componente, en una lista. El acceso y la consulta en esta lista debe ser posible en $\mathcal{O}(1)$. Este tipo de preprocesamiento se puede realizar en $\mathcal{O}(n^2)$.

Notar que la ejecución de la rutina $R([A \rightarrow \beta \cdot, i], j)$ lleva una cantidad de operaciones del orden de $(j-i)^2$; el ciclo lleva $|\beta|$ iteraciones, y en cada iteración lo más costoso es buscar el índice i' examinando $j-i+1$ listas (donde cada inspección se realiza en tiempo constante), y por otro lado hacer la llamada recursiva a $R([\dots, i'], j')$, donde $j' - i' < j - i$.

La ejecución de la rutina $R([A \rightarrow \beta \cdot, i], j)$ invoca recursivamente a $R([\dots, i'], j')$ a lo sumo $|P| \times M$ veces (cada una de ellas con un valor distinto en el ítem), donde M es la máxima cantidad de símbolos de una producción de la gramática.

El total de invocaciones a la rutina R es entonces a lo sumo $(j-i) \times |P| \times M$ (una por cada iteración del ciclo, más las invocaciones recursivas anidadas), lo cual es cuadrático en el tamaño de entrada. □

²²[A&Uv1] Teorema 4.10

²³[A&Uv1] Teorema 4.12

9.3. Fuentes

- Verónica Becher. Teoría de Lenguajes, Clase Teórica 13. Primer cuatrimestre, 2022.
- Verónica Becher. Teoría de Lenguajes, Clase Teórica 14. Primer cuatrimestre, 2022.

10. Gramáticas de Atributos

Las gramáticas de atributos son gramáticas libres de contexto extendidas con *atributos*, *reglas de evaluación* y *condiciones*.

La utilidad de estas extensiones es:

- Proveer sensibilidad al contexto.
- Dar semántica de lenguajes formales.

A cada símbolo de la gramática se le asigna un conjunto de atributos; cada atributo tiene un dominio de valores (e.g. enteros, caracteres, *strings*...). Considerando el árbol de derivación de la gramática libre de contexto base, se puede decir que la gramática de atributos puede pasar valores de un nodo a su ancestro (*atributos sintetizados*) o a sus descendientes (*atributos heredados*).

Más formalmente:

Definición 10.1. Una **gramática de atributos** es una 4-upla $G_A = \langle G, A, V, R \rangle$, donde:

- $G = \langle V_N, V_T, P, S \rangle$ es una gramática libre de contexto **no ambigua** con todos sus símbolos no terminales **alcanzables**.
- A es el conjunto de atributos, y es la unión de los subconjuntos $A(X)$, los atributos asociados al símbolo $X \in V_N \cup V_T$.
- V es el conjunto de dominios de los valores de los atributos.
- R es el conjunto de reglas asociadas a cada producción:

$$X_0.a_0 = f(X_1.a_1, \dots, X_k.a_k)$$

, con $X.a$ denotando el atributo a del símbolo X , y f la función que le asigna valor al atributo a_0 a partir de los valores de los atributos a_1, \dots, a_k .

10.1. Atributos sintetizados y heredados

El conjunto de atributos de un símbolo no terminal X está compuesto de dos subconjuntos: los atributos sintetizados $S(X)$ y los heredados $I(X)$.

Definición 10.2. Sea $G_A = \langle G, A, V, R \rangle$ una gramática de atributos, con $G = \langle V_N, V_T, P, S \rangle$ la gramática libre de contexto.

- Un atributo $X.a$ se dice **sintetizado** si existe una producción $p : X \rightarrow \alpha$ y una regla $r \in R$ para p , donde ocurre $X.a$.
- Un atributo $X.a$ se dice **heredado** si existe una producción $p : Y \rightarrow \alpha X \beta$ y una regla $r \in R$ para p , donde ocurre $X.a$.

Observación. Sobre los atributos sintetizados y heredados:

1. El símbolo distinguido de la gramática G no tiene atributos heredados.

2. No puede haber un atributo que sea sintetizado y heredado simultáneamente, i.e. $S(X) \cap I(X) = \emptyset$.
3. Si $w \in \mathcal{L}(G)$, entonces cada atributo tomará un valor.
4. En el árbol de derivación de una cadena w , el valor de un atributo sintetizado en un nodo se obtiene desde los nodos descendientes, y el valor de un atributo heredado se obtiene de los ancestros o los hermanos.

10.2. Agregando sensibilidad al contexto

Considerar el lenguaje $L = \{a^n b^n c^n : n \geq 1\}$. Se vio previamente que L no es libre de contexto (es dependiente del contexto).

Notar que $L \subset \mathcal{L}(a^+ b^+ c^+)$, un lenguaje regular (se puede expresar como una expresión regular). La gramática libre de contexto $G = \langle \{S, A, B, C\}, \{a, b, c\}, P, S \rangle$ reconoce $\mathcal{L}(a^+ b^+ c^+)$, con producciones:

$$S \rightarrow ABC$$

$$A \rightarrow a \mid Aa$$

$$B \rightarrow b \mid Bb$$

$$C \rightarrow c \mid Cc$$

Utilizando un atributo sintetizado *long* para los símbolos no terminales A, B, C , se puede *controlar* la cantidad de a, b, c que aparecen. En la Figura 15 se puede ver un ejemplo de derivación. Esto mismo se puede conseguir también con atributos heredados.

Producción	Regla
$S \rightarrow ABC$	condición: $A.long = B.long = C.long$
$A \rightarrow a$	$A.long \leftarrow 1$
$A_1 \rightarrow A_2 a$	$A_1.long \leftarrow A_2.long + 1$
$B \rightarrow b$	$B.long \leftarrow 1$
$B_1 \rightarrow B_2 b$	$B_1.long \leftarrow B_2.long + 1$
$C \rightarrow c$	$C.long \leftarrow 1$
$C_1 \rightarrow C_2 c$	$C_1.long \leftarrow C_2.long + 1$

Lo que se consiguió utilizando atributos en este ejemplo, es utilizar una gramática libre de contexto (**extendida**) para generar un lenguaje dependiente del contexto.

Observación. Por lo mostrado recién, se puede afirmar que las gramáticas de atributos tienen mayor poder expresivo que las gramáticas libres de contexto.

Para cada producción $p \in P$, se define el grafo de dependencias D_p : si una regla define el atributo a en función del atributo b (y posiblemente en función de otros atributos), se escribe la arista $X_j.b \rightarrow X_i.a$.

Se define el **grafo de dependencias** D mediante la composición de los grafos D_p .

Si $p \in P$ es de la forma $Y \rightarrow X_1 \dots X_m$, y G_i es un grafo dirigido tal que sus vértices son un subconjunto de $A(X_i)$, entonces $D_p[G_1, \dots, G_m]$ es el grafo que surge de agregar en D_p los ejes de cada G_i , asociándolos a los atributos del símbolo X_i .

Definición 10.6. Una gramática de atributos se dice **circular** si su grafo de dependencias contiene al menos un ciclo.

Para definir el concepto de gramática de atributos bien definida, recordar primero la definición de *orden topológico*:

Definición 10.7. Dado un grafo $G = (V, E)$, un **orden topológico** de G es un orden total de todos sus vértices, de manera tal que cada vértice aparece exactamente una vez, y si $(u, v) \in E$ entonces u precede a v en el orden, para todos $u, v \in V$.

Definición 10.8. Una gramática de atributos está **bien definida** si para todo árbol atribuido, cualquier orden topológico de su grafo de dependencias resulta en un orden de evaluación.

10.3.2. Algoritmo de detección de circularidad

Como ya se mencionó, para poder afirmar que existe un orden de evaluación para los atributos de una gramática (y poder así obtener el significado de una cadena), es necesario verificar que el grafo de dependencias de la gramática no tenga ciclos. Teniendo en cuenta que hay un grafo por cada producción, sería costoso verificar que todos estos grafos sean acíclicos con algún mecanismo convencional.

Para eso se estudia el algoritmo de detección de circularidad estudiado, cuyo funcionamiento es el siguiente:

1. Para cada producción $p \in P$, definir un grafo D'_p , que contiene un vértice por cada atributo asociado a la cabeza de la producción p . La arista (α, β) pertenece al grafo si hay dependencias directas entre α y β en D_p , el grafo de dependencias para la producción.
2. Repetir lo siguiente hasta que ningún D'_p cambie:
 - Para cada producción $p : Y \rightarrow X_1 \dots X_m$ y cada combinación de producciones q_0, \dots, q_m que puedan seguir en la derivación:
 - a) Generar el grafo de $D_p[D'_{q_0}, \dots, D'_{q_m}]$.
 - b) Si para un par de vértices $a, b \in V(D'_p)$ existe un camino entre ellos en ese grafo, entonces agregar la arista (a, b) al grafo D'_p .
3. Si hay algún ciclo en alguno de los D'_p , entonces la gramática es circular.

Al finalizar, se demuestra que se cumple que:

- Si existe una arista (a, b) en un grafo D'_p , es porque hay un camino desde a hasta b en algún árbol de derivación; y eso significa que el algoritmo debe añadir la arista (a, b) a D'_p .
- Para una producción $p : X \rightarrow \alpha$, el grafo D'_p captura todas las posibles dependencias entre atributos de X que puedan surgir en cualquier árbol de derivación parcial con raíz X , que comience con esa producción.

La observación que hace posible este algoritmo es que: si hay un ciclo en algún grafo de dependencias para un árbol atribuido, entonces hay un ciclo simple (que no repite vértices). Entonces, en vez de revisar $2^{m \times m}$ conjuntos de pares de atributos (que sería la cantidad de subconjuntos de dependencias), basta con revisar $2^{m \times \log(m)} = m \times 2^m$ pares, siendo m la cantidad de atributos de la gramática.

Dada una gramática de atributos G_A , el algoritmo produce una gramática libre de contexto G' . G_A es circular cuando una *producción admisible* en G' , que participa en alguna derivación desde el símbolo distinguido de G' hasta una cadena de símbolos terminales, es parte de un ciclo simple (se la llama *arco*).

Gramática equivalente G' : sea $G_A = \langle G, A, V, R \rangle$ la gramática de atributos correspondiente a $G = \langle V_N, V_T, P, S \rangle$; se define la gramática libre de contexto $G' = \langle V'_N, V_T, P', S' \rangle$ de la siguiente manera:

- Para cada producción p de G_A , cada símbolo no terminal X que ocurre en p , y cada subconjunto $D \subseteq \{(a, b) : a, b \in A(p, X)\}$, definir un símbolo no terminal $(X, D) \in V'_N$.
- Para cada producción p de G_A , definir las producciones $Y_0 \rightarrow Y_1 \dots Y_k$:
 - $Y_i \in V'_N \cup V_T$.
 - p es primero(Y_0) \rightarrow primero(Y_1)...primero(Y_k)

Ejemplo 10.1. Dada G_A que contiene $p = X \rightarrow YZ$, con $A(X) = \{a, b, c\}$, $A(Y) = \{d, e, f, g, h\}$, $A(Z) = \{i, j\}$.

Considerando las siguientes dependencias:

$$(a, b), (a, e), (f, b), (f, g), (h, i), (i, j), (j, c),$$

Entonces, los símbolos no terminales definidos para G' serían:

- $(X, \{(a, c)\}) \rightarrow (Y, \{(e, h)\}) (Z, \{(i, j)\})$
- $(X, \{(a, b), (a, c)\}) \rightarrow (Y, \{(e, f), (g, h)\}) (Z, \{(i, j)\})$
- $(X, \{(a, c)\}) \rightarrow (Y, \{(e, f), (e, h), (g, h)\}) (Z, \emptyset)$
- $(X, \{(a, c)\}) \rightarrow (Y, \{(e, f)\}) (Z, \{(i, j)\})$
- $(X, \{(a, b)\}) \rightarrow (Y, \{(d, h)\}) (Z, \emptyset)$

Observación. Sobre G' :

- $|G'| \leq 2^{|G|}$.

- Si G_A es una gramática de atributos, y G' es la gramática libre de contexto definida, se puede probar (por inducción en los árboles de derivación parciales) que (X, D) deriva una cadena de terminales en G' si, y sólo si, X la deriva en G_A y para cada arista (a, b) en D , hay un camino simple desde a hasta b en el grafo de dependencias.

Producciones admisibles: suponer que no existen atributos asociados a símbolos terminales (sólo a no terminales). Sea $\pi \in P'$ una producción de G' , y primero(π) la producción de G_A .

Definición 10.9. Dos atributos e y f (pueden ser iguales) están π -relacionados cuando:

- hay un camino único desde e hasta f en el grafo de dependencias de G_A , o bien
- $(e, f) \in D$ y (X, D) ocurre en el cuerpo de π .

Definición 10.10. Una producción $\pi : (X, D) \rightarrow \dots$ se dice **admisible** si para cada $(a, b) \in D$, vale que $(a, b) \in \pi^+$ (la clausura transitiva de la relación π).

Ejemplo 10.2. Con la información del Ejemplo 10.1, se tiene que:

- $(X, \{(a, c)\}) \rightarrow (Y, \{(e, h)\}) (Z, \{(i, j)\})$ es admisible, porque $(a, c) \in \pi^+$, ya que (e, h) está en el cuerpo de la producción.
- $(X, \{(a, c)\}) \rightarrow (Y, \{(e, f), (g, h)\}) (Z, \{(i, j)\})$ es admisible.
- $(X, \{(a, c)\}) \rightarrow (Y, \{(e, f), (e, h), (g, h)\})$ no es admisible.
- $(X, \{(a, c)\}) \rightarrow (Y, \{(e, f)\}) (Z, \{(i, j)\})$ no es admisible, porque falta la conexión $g \rightarrow h$.
- $(X, \{(a, b)\}) \rightarrow (Y, \{(d, h)\}) (Z, \emptyset)$ es admisible.

Ciclos y arcos: el algoritmo de detección de circularidad identifica las producciones admisibles de G' que forman un *arco* que cierra un ciclo simple. Se concluirá que G_A tiene un ciclo cuando hay alguno de esos arcos.

Definición 10.11. Una producción π es un **arco** si existe al menos un atributo mencionado en π que está π^+ -relacionado consigo mismo.

Ejemplo 10.3. Siguiendo con el Ejemplo 10.1:

- Si en el grafo de dependencias hay un camino desde c hasta a , la producción admisible $(X, \{(a, c)\}) \rightarrow (Y, \{(e, h)\}) (Z, \{(i, j)\})$ es un arco.
- Si en el grafo de dependencias hay un camino desde j hasta g y desde h hasta i , entonces la producción $(X, \emptyset) \rightarrow (Y, \{(g, h)\}) (Z, \{(i, j)\})$ es un arco, debido al ciclo $g \rightarrow \dots \rightarrow h \rightarrow i \rightarrow \dots \rightarrow j \rightarrow g$.

Algoritmo de detección de circularidad

- Entrada: gramática de atributos G_A .
- Salida: decisión sobre la circularidad del grafo de dependencias.
- Procedimiento:

1. Generar los símbolos no terminales de G' .
2. Generar las producciones de G' .
3. Verificar la admisibilidad de las nuevas producciones, revisando el grafo de dependencias de las producciones originales.
4. Remover de G' las producciones inútiles (aquellas que no participan en la derivación de un símbolo terminal, empezando desde el símbolo distinguido).
5. Si en G' hay alguna producción que sea un arco, responder G_A es circular; en caso contrario, responder G_A está bien definida.

Complejidad del algoritmo: en cada número, se describe la complejidad temporal del paso correspondiente del algoritmo:

1. Si G_A tiene m atributos, hay $2^{m \times m}$ posibles subconjuntos de dependencias. Entonces, $|G'|$ es exponencial en $|G_A|$.
2. Este paso se realiza en tiempo exponencial en $|G_A|$.
3. Esto se hace para cada producción, en tiempo exponencial en $|G_A|$.
4. Esto se realiza en tiempo polinomial en $|G'|$.
5. Alcanza con buscar los ciclos simples (por la observación realizada al principio, que se refería a que si hay algún ciclo en uno de los grafos de dependencias para un árbol atribuido, entonces hay un ciclo simple. Esto se realiza, entonces, en tiempo exponencial en G_A).

Teorema 10.1. *La determinación de si una gramática de atributos G_A es circular se puede realizar con complejidad 2^{dn} , siendo n el tamaño de G_A , y d una constante que depende del número máximo de símbolos no terminales que aparecen en el cuerpo de las producciones.*

10.4. Gramáticas s -atribuidas

La clase de gramáticas s -atribuidas es la más restrictiva de las gramáticas de atributos: sólo permite atributos sintetizados.

Definición 10.12. Una gramática de atributos $G_A = \langle G, A, V, R \rangle$ es s -atribuida si el conjunto de atributos heredados $I(A)$ es vacío, y el grafo de dependencias directas de cada producción no tiene ciclos.

Observación. Las gramáticas s -atribuidas se pueden evaluar en un solo recorrido ascendente del árbol atribuido.

Teorema 10.2. *Sea $G_A = \langle G, A, V, R \rangle$ una gramática de atributos bien definida. Existe una gramática de atributos G'_A equivalente a G_A y que no contiene atributos heredados.*

Demostración. Primero, aplicar repetidamente a las producciones las reglas de factorización a izquierda y eliminación de la recursión a izquierda, para finalmente reemplazar los atributos heredados por sintetizados.

Luego, reemplazar cada producción de la forma $X \rightarrow \alpha Y \beta Z \gamma$ (donde un atributo heredado de Y depende de atributos de Z y donde $Y \xrightarrow{*} t$) por una de estas formas:

- Forma 1:

$$\begin{aligned} X &\rightarrow Y'\gamma \\ Y' &\rightarrow \alpha t\beta Z \end{aligned}$$

- Forma 2:

$$\begin{aligned} X &\rightarrow \alpha tY' \\ Y' &\rightarrow \beta Z\gamma \end{aligned}$$

Notar que en las dos formas se cumple que los atributos de Y' no heredan de X ni de Z . \square

10.5. Gramáticas l -atribuidas

La clase de gramáticas l -atribuidas tiene como característica que en cada producción p , los atributos heredados de un símbolo X_j del cuerpo de p , dependen sólo de atributos de los símbolos X_i que aparecen a la izquierda de X_j .

Definición 10.13. Una gramática de atributos $G_A = \langle G, A, V, R \rangle$ es l -atribuida si es bien definida y para cada producción de la forma $p : X_0 \rightarrow X_1 X_2 \dots X_n$ se cumple que:

1. Si la arista $(X_i.a, X_j.b)$ pertenece al grafo de dependencias de p , entonces $0 \leq i < j$.
2. Si la arista $(X_0.a, X_j.b)$ pertenece al grafo de dependencias de p , entonces a es un atributo heredado de X_0 .
3. El grafo de dependencias de p es acíclico.

Observación. Las gramáticas l -atribuidas se pueden evaluar en un solo recorrido descendente, de izquierda a derecha, del árbol atribuido.

10.6. Gramáticas de atributos ordenadas

Definición 10.14. Una gramática de atributos es **ordenada** si para cada símbolo de la gramática, hay un orden sobre los atributos asociados a él, tal que: en cualquier contexto del símbolo, sus atributos se pueden evaluar en dicho orden.

Observación. La verificación de si una gramática de atributos es ordenada se puede realizar en tiempo polinomial en el tamaño de la entrada.

10.7. Traducción Dirigida por Sintaxis

Las traducciones dirigidas por sintaxis (TDS) son otra forma de incorporar análisis semántico al *parsing* de una gramática libre de contexto. Al igual que las gramáticas de atributos, se basan en asociar atributos tipados a los símbolos de la gramática, pero su definición es *de más bajo nivel*.

Se asocia a cada producción distintos fragmentos de código (no sólo asignaciones y condiciones como en las gramáticas de atributos, sino cualquier código que compute algo). Esto posibilita, por ejemplo, imprimir un resultado computado una vez que se obtiene su valor.

Los bloques de código van intercalados con el cuerpo de las producciones, y tienen el mismo tratamiento que los símbolos no terminales en el árbol de derivación. A diferencia de lo que ocurría con las gramáticas de atributos, en las que el árbol atribuido tenía un posible orden de evaluación dado en base a las dependencias de los atributos, en las TDS el orden de *parsing* queda determinado por el orden de las hojas del árbol, de izquierda a derecha. Así, el orden viene dado por la posición de cada bloque de código en la producción, y el programa se puede construir a medida que se arma el árbol sintáctico.

10.8. Fuentes

- Verónica Becher. Teoría de Lenguajes, Clase Teórica 12. Primer cuatrimestre, 2022.
- Natalia Pesaresi. Teoría de Lenguajes, Clase Práctica 15. Primer cuatrimestre, 2022.
- Leonardo Cremona. Teoría de Lenguajes, Clase Práctica 16. Primer cuatrimestre, 2022.

11. Lenguajes Dependientes del Contexto y Lenguajes Recursivos

11.1. Máquinas de Turing

Las máquinas de Turing son un tipo de autómatas que, además de estados y una función de transición entre ellos, tiene una cinta infinita en la que puede escribir y leer con una *cabeza lecto-escritora*. Así, la cabeza puede moverse a la izquierda o a la derecha en la cinta, y puede escribir un símbolo de un alfabeto preestablecido (o colocar un espacio en blanco, que sería el equivalente de *borrar*). La acción dependerá de la posición de la cabeza en la cinta, y de lo que haya escrito en dicha posición.

Al llegar a un estado final, la máquina contesta si la cadena analizada es reconocida o no. Pero a diferencia de lo que pasa con los autómatas vistos antes, en una máquina de Turing es posible que la forma de rechazar algunas entradas sea no detenerse nunca.

Definición 11.1. Una **máquina de Turing** (MT) es un autómata M definido por la siguiente 7-upla $M = \langle Q, \Sigma, \Gamma, \delta, q_0, B, F \rangle$ donde

- Q es el conjunto (finito) de estados.
- Γ es el conjunto (finito) de *símbolos de cinta*.
- $B \in \Gamma$ es el *espacio en blanco*.
- $\Sigma \subset \Gamma$ es el conjunto de símbolos de entrada, con $B \notin \Sigma$.
- $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ es la función de transición.
- $q_0 \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados finales.

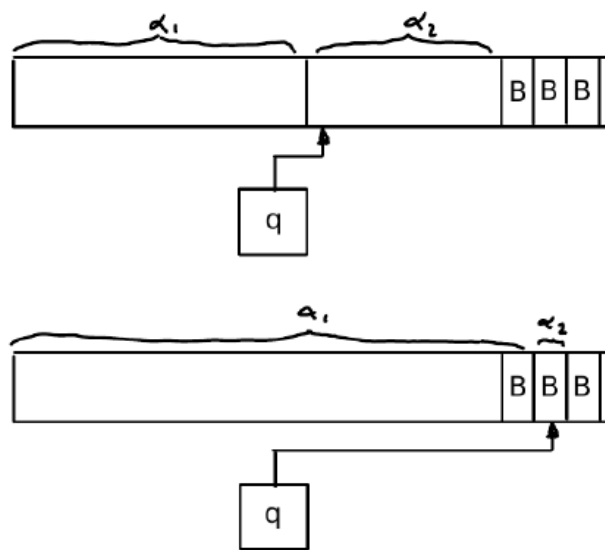


Figura 16: Ejemplo de configuraciones instantáneas $\alpha_1 q \alpha_2$.

Definición 11.2. Una **configuración instantánea** de una máquina de Turing está dada por $\alpha_1 q \alpha_2$, con $q \in Q$, y $\alpha_1, \alpha_2 \in \Gamma^*$, donde:

- α_1 es el contenido de la cinta desde su primera posición hasta la posición que está a la izquierda de la cabeza lecto-escritora.
- α_2 es el contenido de la cinta desde la posición de la cabeza lecto-escritora, hasta la posición del símbolo distinto de B (blanco) más a la derecha en la cinta, o hasta la posición más a la izquierda de la cabeza (la que quede más a la derecha).

11.1.1. Transiciones de una Máquina de Turing

Las transiciones de una máquina de Turing (MT) M son tales que: si M está en una configuración dada por $X_1 \dots X_{i-1} q X_i \dots X_n$ (i.e. con la cabeza lecto-escritora en posición i), entonces es cierto que:

- si $\delta(q, X_i) = (p, Y, L)$ entonces:

- si $1 < i \leq n - 1$,

$$X_1 \dots X_{i-1} q X_i \dots X_n \vdash_M X_1 \dots X_{i-2} p X_{i-1} Y \dots X_n$$

- si $i = n + 1$,

$$X_1 \dots X_n q B \vdash_M X_1 \dots p X_n Y$$

- si $i = 1$, no hay movimiento posible hacia la izquierda.

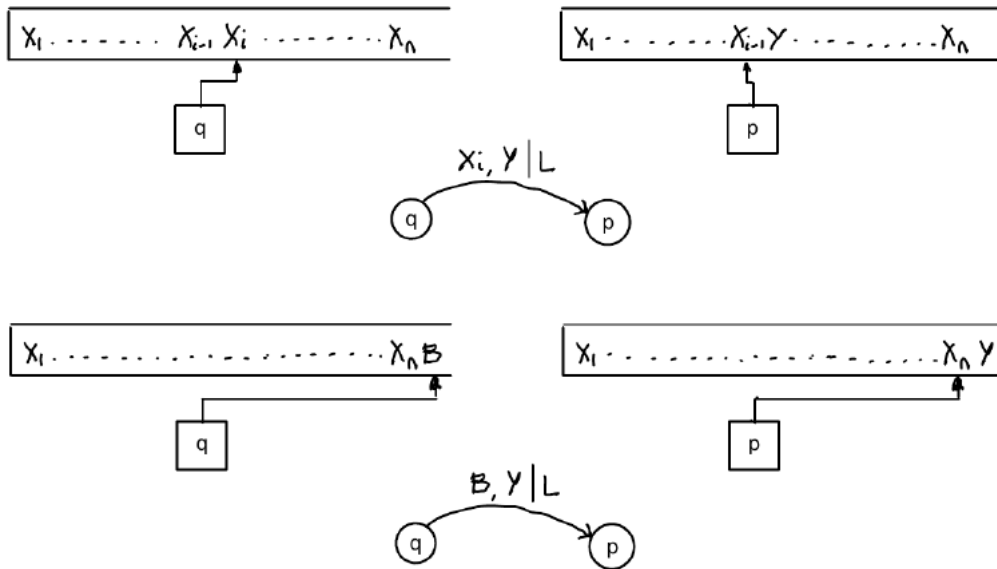


Figura 17: Transiciones a la izquierda.

- si $\delta(q, X_i) = (p, Y, R)$ entonces:

- si $1 \leq i \leq n - 1$,

$$X_1 \dots q X_i X_{i+1} \dots X_n \vdash_M X_1 \dots Y p X_{i+1} \dots X_n$$

- si $i = n + 1$,

$$X_1 \dots X_n q B B \vdash_M X_1 \dots X_n Y p B$$

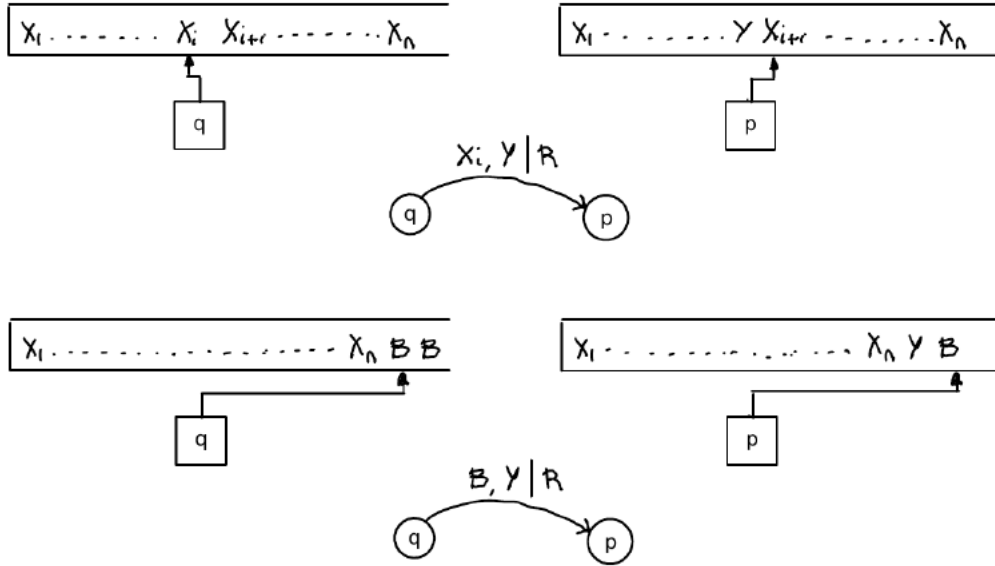


Figura 18: Transiciones a la derecha.

Definición 11.3. El lenguaje aceptado por una máquina de Turing M se define como

$$\mathcal{L}(M) = \{w \in \Sigma^* : q_0 w \xrightarrow{*}_M \alpha_1 p \alpha_2, \text{ con } p \in F, \text{ y } \alpha_1, \alpha_2 \in \Gamma^*\}$$

11.1.2. Equivalencia entre Máquinas de Turing Determinísticas y No Determinísticas

Al igual que ocurría con los autómatas finitos (pero no con los autómatas de pila), las máquinas de Turing determinísticas reconocen los mismos lenguajes que las no determinísticas.

Teorema 11.1. Sea M una máquina de Turing no determinística (MTND), y sea L el lenguaje reconocido por la misma, i.e. $L = \mathcal{L}(M)$. Entonces, existe una máquina de Turing determinística (MTD) M' tal que $L = \mathcal{L}(M')$.

Demostración. Tener en cuenta:

- Para cada estado de la máquina M , existe una cantidad de transiciones que parten de él. Sea r la máxima cantidad posible de transiciones que parten de un estado cualquiera en M .
- Para cada estado, se numeran las transiciones que parten de él, con números entre 1 y r .
- Toda secuencia finita de enteros entre 1 y r puede ser interpretada entonces como una secuencia de transiciones en la MT M , partiendo desde el estado inicial q_0 . Notar que algunas de estas secuencias no serán ejecutables, por no existir algunas de sus transiciones, para algunos de los estados.

Sea M' una máquina de Turing con tres cintas:

- La primera cinta contendrá la cadena de entrada, la cual no será alterada.
- La segunda cinta contendrá una secuencia de enteros entre 1 y r , que corresponderá a una secuencia de transiciones a partir del estado inicial q_0 .

- La tercera cinta se utilizará para simular la MTND M .

El funcionamiento de M' será el siguiente:

- Se generan secuencias de enteros entre 1 y r , en orden creciente según longitud, y en orden alfabético. Estas secuencias se irán escribiendo en la cinta 2. Para cada una de ellas, se ejecuta los siguientes pasos hasta aceptar la cadena de la cinta 1:
 - Borrar el contenido de la cinta 3.
 - Copiar el contenido de la cinta 1 en la cinta 3.
 - Simular la MT M en la cinta 3, ejecutando la secuencia de transiciones en curso especificada en la cinta 2.
 - Aceptar la cadena de la cinta 1 si se acepta la misma cadena en la simulación corriendo en la cinta 3.
- Cuando la cadena analizada no es aceptada por la MT M , entonces la M' no se detendrá.

□

11.1.3. Codificación de Máquinas de Turing restringidas a un alfabeto binario

Sea $M = \langle Q, \{0, 1\}, \{0, 1, B\}, \delta, q_1, B, \{q_2\} \rangle$ una máquina de Turing, con $Q = \{q_1, q_2, \dots, q_n\}$ su conjunto de estados.

Si se nombra a los símbolos 0, 1 y B como X_1 , X_2 y X_3 respectivamente, y a los símbolos L y R como D_1 y D_2 respectivamente, se puede codificar cada transición $\delta(q_i, X_j) = (q_k, X_\ell, D_m)$, con $1 \leq i, k \leq n, 1 \leq j, \ell \leq 3, 1 \leq m \leq 2$ como:

$$0^i 10^j 10^k 10^\ell 10^m 1$$

Y el código binario para la máquina de Turing M será entonces:

$$111 \text{ trans}_1 11 \dots 11 \text{ trans}_r 111$$

, donde los trans_i son las codificaciones para las transiciones.

11.2. Autómatas Linealmente Acotados

Un autómata lineal acotado es como una máquina de Turing, pero la cinta no es infinita, sino que su longitud es igual a la longitud de la cadena de entrada (o tiene valor que depende linealmente de dicha longitud).

Definición 11.4. Un **autómata linealmente acotado** (ALA) es una 8-upla $\langle M = Q, \Sigma, \Gamma, \delta, q_0, c, \$, F \rangle$, donde:

- Q es el conjunto (finito) de estados.
- Γ es el conjunto (finito) de símbolos de cinta.
- $\Sigma \subset \Gamma$ es el conjunto de símbolos de entrada.

- $\epsilon, \$ \in \Gamma$ son símbolos especiales.
- δ es la función de transición.
- $q_0 \in Q$ es el estado inicial.
- $F \subseteq Q$ es el conjunto de estados finales.

Este autómata es similar a una máquina de Turing no determinística (MTND), pero que tiene las siguientes condiciones:

- El alfabeto de entrada incluye dos símbolos $\epsilon, \$$, utilizados como *topes* izquierdo y derecho, respectivamente.
- El ALA no se mueve a la izquierda de ϵ , ni a la derecha de $\$$, y tampoco los sobrescribe.

Definición 11.5. El **lenguaje aceptado** por un autómata linealmente acotado M se define como:

$$\mathcal{L}(M) = \{w : w \in (\Sigma \setminus \{\epsilon, \$\})^* \wedge q_0 \epsilon w \$ \stackrel{*}{\vdash}_M \alpha q \beta, \text{ para } q \in F\}$$

11.3. Lenguajes Dependientes del Contexto

Recordando la definición de una gramática dependiente del (o sensitiva al) contexto:

Definición 11.6. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática. Se dice que G es **dependiente del contexto** (GDC) si todas sus producciones son de la forma $\alpha \rightarrow \beta$, donde $\alpha, \beta \in (V_N \cup V_T)^*$ y $|\alpha| \leq |\beta|$.

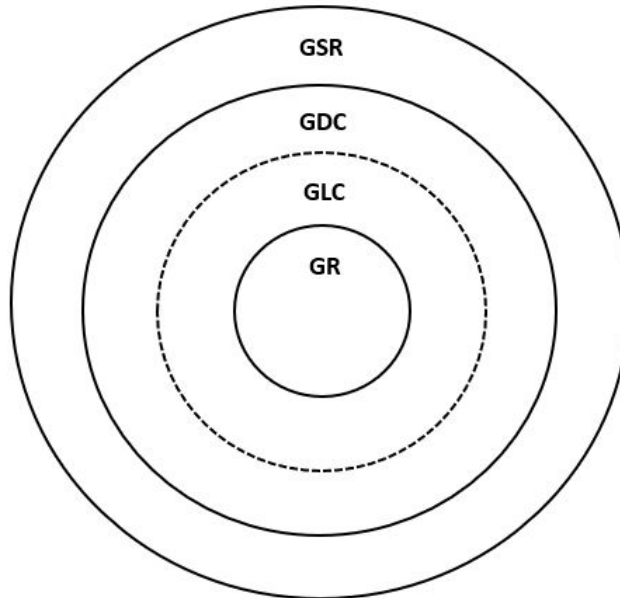


Figura 19: Jerarquía de gramáticas (GSR: Gramáticas Sin Restricciones; GDC: Gramáticas Dependientes del Contexto; GLC: Gramáticas Libres de Contexto; GR: Gramáticas Regulares)

Observación. Por definición, las gramáticas dependientes del contexto no pueden tener *reglas borradoras*: no es posible generar la cadena nula λ .

A partir de esto, se puede deducir que: *toda GLC* (gramática libre de contexto) *sin reglas borradoras* (es decir, reglas de tipo $A \rightarrow \lambda$), **también es una GDC**.

Entonces, se puede establecer la jerarquía de gramáticas de la Figura 19 (jerarquía de Chomsky). La línea que separa a las gramáticas libres de contexto de las gramáticas dependientes del contexto es punteada por la observación previa.

11.3.1. Relación entre Gramáticas Dependientes del Contexto y Autómatas Linealmente Acotados

Teorema 11.2. *Sea L un lenguaje dependiente del contexto (es decir, $L = \mathcal{L}(G)$, con G una gramática dependiente del contexto). Entonces existe un autómata linealmente acotado (ALA) M tal que $L = \mathcal{L}(M)$.*

Demostración. Sea $G = \langle V_N, V_T, P, S \rangle$ la gramática que genera L . Construyendo una MTND (máquina de Turing no determinística) de dos cintas:

- La primera cinta contiene la cadena de entrada $cw\$$, que permanece inalterada.
- La segunda cinta se utiliza para generar las formas sentenciales de la derivación. En cualquier instante, esta cinta contendrá la forma sentencial α de la derivación de la cadena de entrada. Se inicializa con el símbolo distinguido S .

El ALA M operará de la siguiente manera:

1. Si $w = \lambda$, entonces M se detiene, rechazando la cadena de entrada.
2. Seleccionar (en forma no determinística) la posición i dentro de α .
3. Seleccionar (en forma no determinística) la producción $\beta \rightarrow \gamma \in P$.
4. Si β aparece a partir de la posición i en α , entonces reemplazar β por γ en α .
5. Si la nueva forma sentencial α es tal que $|\alpha| > |w|$, entonces M se detiene, rechazando la cadena de entrada.
6. Comparar la nueva forma sentencial α resultante con la cadena de entrada w . Si $\alpha = w$, entonces aceptar w ; si no, volver al paso 2.

□

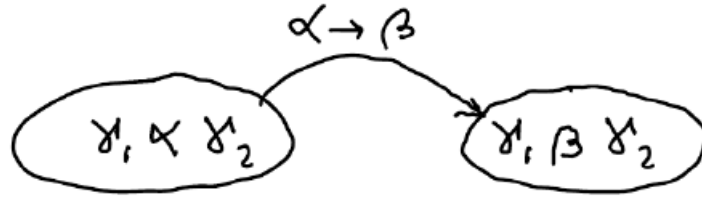
11.3.2. Relación entre Lenguajes Dependientes del Contexto y Lenguajes Recursivos

Teorema 11.3. *Todo lenguaje dependiente del contexto L es un lenguaje recursivo. Es decir, existe un algoritmo que permite decidir la pertenencia de toda cadena al lenguaje L .*

Demostración. Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática dependiente del contexto tal que $L = \mathcal{L}(G)$. Se formulará un algoritmo que determina, para toda cadena $w \in V_T^*$, si $w \in \mathcal{L}(G)$ o no.

Para eso, primero se construye un grafo finito, en el cual existe un vértice por cada cadena (de símbolos terminales o no terminales, es decir en $(V_N \cup V_T)^+$), de longitud entre 1 y $|w|$. Se coloca una arista entre dos vértices cuando $\gamma_1\alpha\gamma_2 \xRightarrow{G} \gamma_1\beta\gamma_2$, donde $\gamma_1\alpha\gamma_2$ y $\gamma_1\beta\gamma_2$ son las cadenas correspondientes al vértice de origen y de destino, respectivamente; y donde $\alpha \rightarrow \beta \in P$.

Entonces, para la cadena w se tendrán vértices y aristas como los que se ven en la siguiente imagen:



En dicha imagen se puede observar que $\gamma_1, \gamma_2, \beta \in (V_N \cup V_T)^*$, $\alpha \in ((V_N \cup V_T)^* V_N (V_N \cup V_T)^*)$. Además, $|\gamma_1 \alpha \gamma_2| \leq |w|$, $|\gamma_1 \beta \gamma_2| \leq |w|$, y $\alpha \rightarrow \beta \in P$, con $|\alpha| \leq |\beta|$.

Como $|\alpha| \leq |\beta|$, entonces $|\gamma_1 \alpha \gamma_2| \leq |\gamma_1 \beta \gamma_2|$.

Así, en el grafo correspondiente a la gramática G , es cierto que existe un camino desde el vértice correspondiente a S hasta el vértice correspondiente a w , si y sólo si $S \xrightarrow{*}_G w$. Como el grafo es finito, determinar si hay un camino entre dos vértices del mismo es decidible; es decir, hay un algoritmo que determina si eso ocurre o no. Luego, también hay un algoritmo que decide si la cadena w se deriva en G . Por último, como L es el lenguaje generado por G , se prueba que existe un algoritmo que decide la pertenencia de una cadena cualquiera al lenguaje L . \square

11.4. Lenguajes Recursivos y Recursive Enumerables

Mediante un argumento de diagonalización (como el utilizado en la demostración del teorema de Cantor), se puede ver lo siguiente:

Lema 11.1. *Sea M_1, M_2, \dots una enumeración de un conjunto de máquinas de Turing que se detienen para todas las entradas. Entonces, existe un lenguaje recursivo L que no es aceptado por ninguna de ellas. Es decir, siempre existe L tal que:*

- L es recursivo, y
- $\forall j \geq 1 : L \neq \mathcal{L}(M_j)$.

Demostración. Considerar el lenguaje $L \subseteq \{0, 1\}^*$, definido por

$$L = \{w_i : w_i \notin \mathcal{L}(M_i)\}$$

Es decir, L está formado por todas las cadenas w_i que son rechazadas por la máquina con igual índice, M_i . Este lenguaje es recursivo, ya que para toda cadena w_i , vale que: w_i está en L si y sólo si es rechazada por la MT M_i ; y esto es decidible, ya que las máquinas M_i se detienen para toda entrada, por hipótesis. Luego, se puede decidir siempre si una cadena pertenece o no al lenguaje L .

Suponer que el lenguaje L es reconocido por alguna de las MT de la enumeración. Sea M_j dicha máquina, es decir que $L = \mathcal{L}(M_j)$. Considerar entonces el caso de la cadena w_j (cuyo índice coincide con el de la máquina en cuestión). La pregunta es: ¿ $w_j \in \mathcal{L}(M_j)$ o $w_j \notin \mathcal{L}(M_j)$?

Utilizando la definición de L , y que $L = \mathcal{L}(M_j)$, se tiene que:

$$w_j \in L \Leftrightarrow w_j \notin \mathcal{L}(M_j) \Leftrightarrow w_j \notin L$$

Esto es claramente una contradicción, que proviene de considerar que existe un índice $j \geq 1$ tal que $L = \mathcal{L}(M_j)$. Por lo tanto, L es un lenguaje recursivo que no es ninguno de los aceptados por las máquinas de Turing de la enumeración M_1, M_2, \dots . \square

Lema 11.2. *Existe un lenguaje recursivo que no es dependiente del contexto.*

Demostración. La idea será hallar una enumeración de máquinas de Turing, correspondientes a cada uno de los lenguajes dependientes del contexto definidos sobre $\{0, 1\}^*$. Estas MT se detienen en todas las entradas, ya que siempre hay un algoritmo de reconocimiento cuando el lenguaje es dependiente del contexto (por el Teorema 11.3).

Se codificarán las gramáticas dependientes del contexto a través de cadenas binarias (a cada posible símbolo, como 0, 1, \rightarrow , $\{$, símbolos no terminales, etc. le corresponde una cadena de ceros y unos). De esta manera, es posible enumerar estas gramáticas: G_1, G_2, \dots

Como ya se dijo, existe un algoritmo que permite obtener una MT que se detiene en todas las entradas, a partir de una gramática dependiente del contexto. Aplicando este algoritmo a cada una de las gramáticas de la enumeración G_1, G_2, \dots , se obtiene una enumeración M_1, M_2, \dots de máquinas de Turing que se detienen para toda entrada, tales que $\forall i, \mathcal{L}(M_i) = \mathcal{L}(G_i)$.

Luego, utilizando el Lema 11.1 sobre la enumeración M_1, M_2, \dots , se puede afirmar que existe un lenguaje recursivo L dado por

$$L = \{w_i : w_i \notin \mathcal{L}(M_i) = \mathcal{L}(G_i), \text{ con } G_i \text{ una GDC sobre } \{0, 1\}^*\}$$

Como L no es reconocido por ninguna de las máquinas de Turing de la enumeración, y esas máquinas se obtuvieron a partir de una enumeración de todas las gramáticas dependientes del contexto, entonces el lenguaje L no es dependiente del contexto. \square

11.4.1. Relación entre Gramáticas Sin Restricciones y Máquinas de Turing

Teorema 11.4. *Sea $G = \langle V_N, V_T, P, S \rangle$ una gramática sin restricciones (GSR) tal que $L = \mathcal{L}(G)$. Entonces existe una máquina de Turing (MT) M tal que $L = \mathcal{L}(M)$.*

Demostración. Se construirá una MTND de dos cintas:

- La primera cinta contendrá la cadena de entrada w .
- La segunda cinta contendrá la forma sentencial α de la derivación de la cadena de entrada. Se inicializará con el símbolo S .

El funcionamiento de M será el siguiente:

1. Seleccionar (en forma no determinística) la posición i dentro de α .
2. Seleccionar (en forma no determinística) la producción $\beta \rightarrow \gamma \in P$.
3. Si β aparece a partir de la posición i en α , reemplazar β por γ en α .
4. Comparar la nueva forma sentencial α resultante con la cadena de entrada w . Si $\alpha = w$, entonces se acepta w ; si no, se vuelve al paso 1.

\square

Con esta idea se prueba que todo lenguaje generado por una GSR puede ser reconocido por una MTND. Luego, por el Teorema 11.1, se tiene que

$$\text{GSR} \Rightarrow \text{MT}$$

Ahora se verá la propiedad recíproca:

Teorema 11.5. Sea $M = \langle Q, \Sigma, \Gamma, \delta, q_0, B, F \rangle$ una máquina de Turing (MT) tal que $L = \mathcal{L}(M)$. Entonces existe una gramática sin restricciones (GSR) G tal que $L = \mathcal{L}(G)$.

Demostración. La idea será que G genere dos copias de alguna representación de la cadena de entrada, y que luego simule la operación de la máquina M sobre una de ellas.

Si esta simulación resulta en una aceptación de la cadena de entrada, entonces la primera copia (que sigue siendo una representación inalterada de la cadena de entrada) se convierte en la cadena igual a la de entrada.

La gramática $G = \langle V_N, \Sigma, P, A_1 \rangle$ (notar que $V_T = \Sigma$), con $V_N = ((\Sigma \cup \{\lambda\}) \times \Gamma) \cup \{A_1, A_2, A_3\}$, tiene un conjunto de producciones P definido como:

1. $A_1 \rightarrow q_0 A_2$
2. $A_2 \rightarrow [a, a] A_2$, para cada $a \in \Sigma$.
3. $A_2 \rightarrow A_3$
4. $A_3 \rightarrow [\lambda, B] A_3$
5. $A_3 \rightarrow \lambda$
6. $q[a, X] \rightarrow [a, Y] p$, para todo $a \in \Sigma \cup \{\lambda\}$, $q \in Q$, y $X, Y \in \Gamma$ tales que $\delta(q, X) = (p, Y, R)$
7. $[b, Z] q[a, X] \rightarrow p[b, Z][a, Y]$, para todo $a, b \in \Sigma \cup \{\lambda\}$, $q \in Q$, y $X, Y, Z \in \Gamma$ tales que $\delta(q, X) = (p, Y, L)$
8. Para todo $a \in \Sigma \cup \{\lambda\}$, $q \in F$ y $X \in \Gamma$:
 - $[a, X] q \rightarrow qaq$
 - $q[a, X] \rightarrow qaq$
 - $q \rightarrow \lambda$

Utilizando las reglas 1 y 2, se puede generar:

$$A_1 \xrightarrow[G]{*} q_0 [a_1, a_1] \dots [a_n, a_n] A_2$$

Luego, utilizando la regla 3, se generan los símbolos correspondientes a espacios en blanco necesarios para el análisis de la cadena de entrada en la MT M :

$$A_1 \xrightarrow[G]{*} q_0 [a_1, a_1] \dots [a_n, a_n] [\lambda, B]^m A_3$$

Y utilizando las reglas 6 y 7, se simula la operación de M sobre las segundas componentes, dejando inalteradas las primeras componentes.

Sean $a_1, \dots, a_n \in \Sigma$, $a_{n+1} = \dots = a_{n+m} = \lambda$, $X_1, \dots, X_{n+m} \in \Gamma$, y $X_{s+1} = \dots = X_{n+m} = B$. Puede demostrarse que si

$$q_0 a_1 \dots a_n \xrightarrow[M]{*} X_1 \dots X_{r-1} q X_r \dots X_s$$

, entonces:

$$q_0 [a_1, a_1] \dots [a_n, a_n] [\lambda, B]^m \xrightarrow[G]{*} [a_1, X_1] \dots [a_{r-1}, X_{r-1}] q [a_r, X_r] \dots [a_{n+m}, X_{n+m}]$$

Lo que se busca probar es cierto (por antecedente y consecuente verdaderos) cuando la cantidad de transiciones en M es cero, ya que en ese caso $r = 1$ y $s = n$, con lo cual se tiene que si:

$$q_0 a_1 \dots a_n \stackrel{0}{\vdash}_M q a_1 \dots a_n$$

, entonces:

$$q_0 [a_1, a_1] \dots [a_n, a_n] [\lambda, B]^m \stackrel{*}{\Rightarrow}_G q_0 [a_1, a_1] \dots [a_n, a_n] [\lambda, B]^m$$

Así que eso sirve como caso base para una inducción en la cantidad de transiciones. Para el paso inductivo, tomar el caso de k transiciones:

$$q_0 a_1 \dots a_n \stackrel{k-1}{\vdash}_M X_1 \dots X_{r-1} q X_r \dots X_s \stackrel{}{\vdash}_M Y_1 \dots Y_{t-1} p Y_t \dots Y_u$$

Por hipótesis inductiva, se tiene que:

$$q_0 [a_1, a_1] \dots [a_n, a_n] [\lambda, B]^m \stackrel{*}{\Rightarrow}_G [a_1, a_1] \dots [a_{r-1}, X_{r-1}] q [a_r, X_r] \dots [a_{n+m}, X_{n+m}]$$

Si en el paso k el movimiento de M es hacia la derecha, es decir $\delta(q, X_r) = (p, Y_r, R)$, entonces $t = r + 1$; y por la regla 6, se sabe que $q [a_r, X_r] \rightarrow [a_r, Y_r] p \in P$. Por lo tanto:

$$[a_1, a_1] \dots [a_{r-1}, X_{r-1}] q [a_r, X_r] \dots [a_{n+m}, X_{n+m}] \stackrel{*}{\Rightarrow}_G [a_1, a_1] \dots [a_r, Y_r] p [a_{r+1}, X_{r+1}] \dots [a_{n+m}, X_{n+m}]$$

Si en el paso k el movimiento de M es hacia la izquierda, es decir $\delta(q, X_r) = (p, Y_r, L)$, entonces $t = r - 1$; y por la regla 7, se sabe que $[a_{r-1}, X_{r-1}] q [a_r, X_r] \rightarrow p [a_{r-1}, X_{r-1}] [a_r, Y_r] \in P$. Por lo tanto:

$$[a_1, a_1] \dots [a_{r-1}, X_{r-1}] q [a_r, X_r] \dots [a_{n+m}, X_{n+m}] \stackrel{*}{\Rightarrow}_G [a_1, a_1] \dots p [a_{r-1}, X_{r-1}] [a_r, Y_r] \dots [a_{n+m}, X_{n+m}]$$

Luego, si se llega a una forma sentencial en la que el estado q sea final, entonces aplicando repetidas veces la regla 8, se tiene:

$$[a_1, Y_1] \dots [a_{t-1}, Y_{t-1}] q [a_t, Y_t] \dots [a_{n+m}, Y_{n+m}] \stackrel{*}{\Rightarrow}_G a_1 a_2 \dots a_n$$

Con esto queda probado que si $x \in \mathcal{L}(M)$, entonces $x \in \mathcal{L}(G)$. Análogamente, se puede demostrar que vale el sentido inverso, es decir que si $x \in \mathcal{L}(G)$ entonces $x \in \mathcal{L}(M)$. \square

Observación. A partir de los Teoremas 11.4 y 11.5, se puede concluir que las máquinas de Turing (MT) aceptan los mismos lenguajes que generan las gramáticas sin restricciones (GSR), es decir:

$$\text{GSR} \Leftrightarrow \text{MT}$$

11.4.2. Existencia de un lenguaje no recursivamente enumerable

Teorema 11.6. *Existe un lenguaje que no es recursivamente enumerable.*

Demostración. Considerar el ordenamiento del conjunto $\{0, 1\}^*$, primero por longitud y luego por orden lexicográfico. Sea w_i la i -ésima cadena según tal ordenamiento, y sea M_j la máquina de Turing cuyo código en binario (según la codificación vista en 11.1.3) es el número entero positivo j .

Se puede construir entonces una tabla, que indique con un 1, en la posición i, j que $w_i \in \mathcal{L}(M_j)$, y con un 0 cuando no. La tabla tendría esta forma:

		$j \longrightarrow$				
		1	2	3	4
	$i \downarrow$					
1		0	1	1	0
2		1	1	0	0
3		0	0	1	0
4		0	1	0	1
⋮		⋮	⋮	⋮	⋮	⋮
⋮		⋮	⋮	⋮	⋮	⋮
⋮		⋮	⋮	⋮	⋮	⋮

Sea L_d un lenguaje definido como

$$L_d = \{w_i : w_i \notin \mathcal{L}(M_i)\}$$

Es decir, L_d corresponde a los ceros de la diagonal en la tabla anterior. Si L_d es recursivamente enumerable, entonces existe una máquina de Turing M_j tal que $L_d = \mathcal{L}(M_j)$, pero entonces:

$$w_j \in L_d \Leftrightarrow w_j \notin \mathcal{L}(M_j) \Leftrightarrow w_j \notin L_d$$

Esto es una contradicción, que proviene de suponer que L_d es recursivamente enumerable. Por lo tanto, la máquina de Turing M_j no existe, y el lenguaje no es recursivamente enumerable. \square

11.4.3. Existencia de un lenguaje no recursivo

Para demostrar que existe un lenguaje no recursivo (pero sí recursivamente enumerable), se probará primero la existencia de un lenguaje recursivamente enumerable: el **lenguaje universal**.

Definición 11.7. Sea L_u el lenguaje universal, constituido por todos los pares compuestos de una máquina de Turing M , y una cadena de entrada w , tales que la cadena w es aceptada por la máquina M . Es decir:

$$L_u = \{\langle M, w \rangle : w \in \mathcal{L}(M)\}$$

Teorema 11.7. *El lenguaje universal L_u es recursivamente enumerable.*

Demostración. Se construirá una máquina de Turing de tres cintas, que aceptará L_u :

- La primera cinta contendrá la cadena de entrada (es decir, la codificación de la máquina de Turing M , seguida de la cadena w).
- La segunda cinta se utiliza para simular la cinta de la MT original M .
- La tercera cinta contiene el estado actual de la MT simulada.

El funcionamiento de esta máquina será el siguiente:

1. Verificar que la cadena que codifica la MT M en la cinta 1 sea sintácticamente correcta y determinística.
2. Inicializar la cinta 2 con la cadena w , y la cinta 3 con 0, que representa el estado inicial q_0 . Todas las cabezas lecto-escritoras deben posicionarse en la posición más a la izquierda de su cinta.
3. Si el contenido de la cinta 3 es 00 (i.e. 0^2 , correspondiente al estado q_2), entonces terminar y aceptar.
4. Sea X_j el símbolo actualmente leído en la cinta 2, y sea 0^i el contenido de la cinta 3 (codificación correspondiente al estado actual q_i). Entonces:
 - a) Comenzando en la posición más a la izquierda, buscar en la cinta 1 la cadena 110^i10^j1 (correspondiente a $\delta(q_i, X_j)$), hasta encontrarla o hasta hallar 111 (lo que suceda primero).
 - b) Si se encontró la cadena 111, entonces detenerse y rechazar.
 - c) Si se encontró la cadena 110^i10^j1 , entonces existe la transición (codificada) $\delta(q_i, X_j) = (q_k, X_\ell, D_m)$, es decir $0^i10^j10^k10^\ell10^m$. Luego, colocar 0^k (correspondiente al estado q_k) en la cinta 3, escribir X_ℓ en la cinta 2, y moverse en la dirección D_m . Por último, ir al paso 3.

Esta máquina (cuyo comportamiento se puede ver en la Figura 20), acepta el lenguaje L_u ; por lo tanto, el lenguaje es recursivamente enumerable. \square

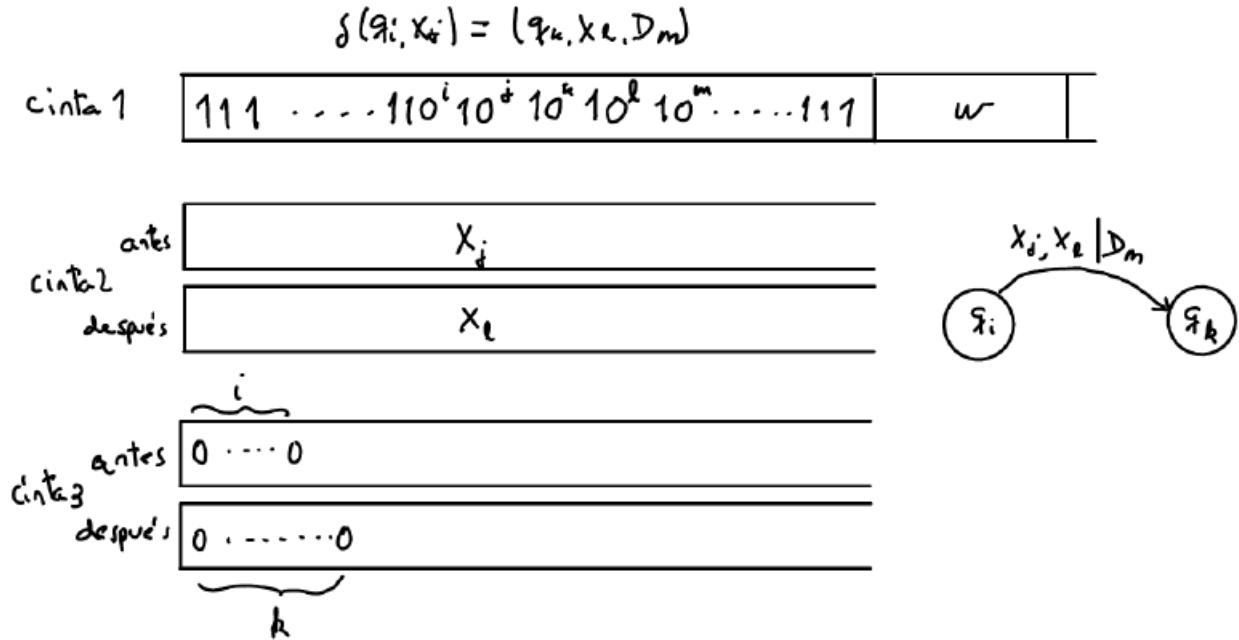
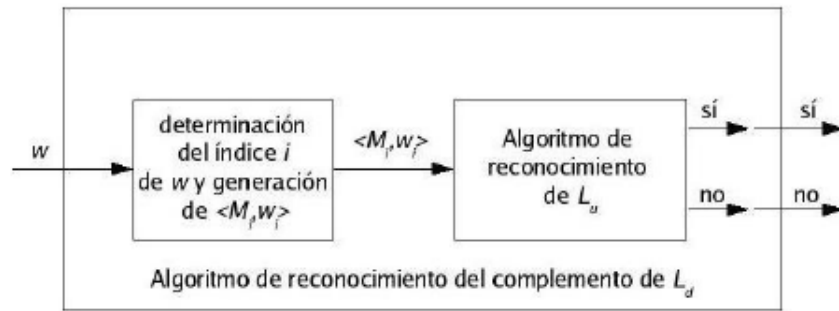


Figura 20: Máquina de Turing M de tres cintas, que reconoce el lenguaje universal.

Teorema 11.8. *El lenguaje universal L_u no es recursivo.*

Demostración. Suponer que L_u sí es recursivo. Entonces, existe un algoritmo para decidir si una cadena pertenece a él. Por lo tanto, se debe poder construir un algoritmo para reconocer el complemento del lenguaje L_d (definido en el Teorema 11.6), de la siguiente manera:



Pero entonces también existiría un algoritmo para reconocer L_d , lo cual es imposible, ya que se vio en el Teorema 11.6 que ese lenguaje no es recursivamente enumerable. Esta contradicción viene de suponer que existe un algoritmo para reconocer L_u . Luego, L_u no es recursivo. \square

Corolario. La consecuencia de los resultados anteriores es que la jerarquía de lenguajes tiene la siguiente forma:

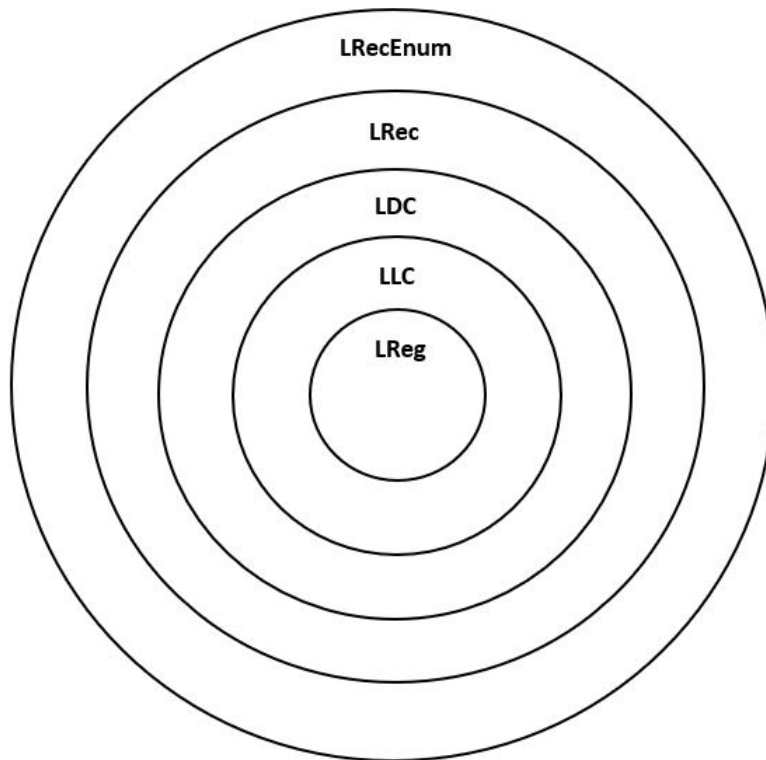


Figura 21: Jerarquía de lenguajes (LRecEnum: Lenguajes Recursivamente Enumerables; LRec: Lenguajes Recursivos; LDC: Lenguajes Dependientes del Contexto; LLC: Lenguajes Libres de Contexto; LReg: Lenguajes Regulares)

11.5. Fuentes

- Julio Jacobo. Teoría de Lenguajes, Clase Teórica 7. Primer cuatrimestre, 2022.

Referencias

- [1] : [A&Uv1] Alfred Aho, Jeffrey Ullman. *The Theory of Parsing, Translation and Compiling*, Volumen 1. Prentice Hall, 1972.
- [2] : [A&Uv2] Alfred Aho, Jeffrey Ullman. *The Theory of Parsing, Translation and Compiling*, Volumen 2. Prentice Hall, 1972.
- [3] : [H&U] John Hopcroft, Rajeev Motwani, Jeffrey Ullman. *Introduction to Automata Theory, Languages and Computation*. Segunda edición, Addison Wesley, 2001.